Escuela Técnica Superior de Ingeniería Naval y Oceánica

Universidad Politécnica de Cartagena

UNIVERSIDAD POLITÉCNICA DE CARTAGENA.

Escuela Técnica Superior de Ingeniería Naval y Oceánica

Métodos Numéricos en el Ámbito Naval.

Autor: Jesús Monserrat Torrecillas.

Profesor: Juan Carlos Trillo Moya.

Departamento de Matemática Aplicada y Estadística

Agradecimientos.

Al Doctor D.Juan Carlos Trillo Moya, por compartir conmigo todo su conocimiento y la gran ayuda prestada durante el desarrollo de este proyecto final de carrera.

Al departamento de Matemática Aplicada y Estadística y a la Universidad Politécnica de Cartagena.

A mis padres y hermana, por su esfuerzo y sacrificio durante todos los años de carrera en los que me han acompañado y ayudado, en especial durante la realización de este proyecto.

Sinopsis.

Durante los últimos años se han dedicado muchos esfuerzos al estudio y perfeccionamiento de métodos numéricos debido a la gran importancia que han adquirido en el diseño industrial además de otros campos de investigación tanto científicos como tecnológicos, desde campos como el biotecnológico hasta la meteorología.

Esta gran aplicación que han adquirido la aplicación los métodos numéricos se debe a que, mediante algoritmos altamente sofisticados, nos permiten realizar aproximaciones y obtener resultados para casi cualquier tipo de problemas que, de otra manera nos sería muy difícil incluso imposible de obtener. Además, permiten un estudio elaborado del error cometido por cada método y en cada caso para así poder tenerlo en cuenta a la hora de la toma de decisiones críticas en el proceso de diseño de ingeniería.

Este documento presenta una serie de métodos numéricos junto con su implementación en Matlab mediante una interfaz gráfica. De esta manera, el usuario es capaz de seleccionar de método que desee o necesite realizar, introducir de un modo sencillo e intuitivo los datos que necesite en la interfaz y obtener los resultados calculados de un modo visiblemente agradable.

Además, a lo largo de éste documento se justificará el uso de métodos en el ámbito naval. Lo cual encuentra aplicación en muchas de sus diversas ramas de estudio, desde el cálculo de estructuras navales hasta el cálculo de la estabilidad hidrostática del buque, pasando por la alineación y suavizado de las formas del casco.

Comenzaremos dicho proyecto enunciando unos conceptos básicos de los métodos numéricos así como de la serie de errores que podemos cometer en éstos y que deberemos tener en cuenta. A continuación se expondrá de forma detallada cada uno de los métodos que hemos considerado divididos en distintos temas según su aplicación, comenzando por la resolución de ecuaciones no lineales, continuamos con resolución de sistemas lineales, interpolación numérica, derivación e integración numérica, acabando por el estudio de métodos de aproximación de ecuaciones diferenciales de primer orden.

Al ser un proyecto de final de carrera de la titulación de grado en arquitectura naval e ingeniería de sistemas marinos, en cada uno de los capítulos se tratará de resolver un caso práctico mediante alguno de los métodos explicados en el capítulo correspondiente referente al diseño naval.

Abstract.

In recent years much effort has been devoted to the study and development of numerical methods because of the great importance acquired in industrial design and other research fields both scientific and technological, from fields such as biotechnology to meteorology.

This great application that have acquired the application of numerical methods is that, using highly sophisticated algorithms, allow us to approach and get results for almost any type of problems that otherwise it would be very difficult if not impossible to obtain. They also allow a study of the error made by each method and in each case in order to take this into account when making critical decisions in the process of engineering design.

This document presents a series of numerical methods with its implementation in Matlab using a graphical interface. In this way, the user is able to select method he wants or needs to perform, introducing the data he needs in a simple and intuitive way in the interface and showing the results calculated in a visibly pleasant mode.

In addition, throughout this document using methods in the naval sphere will be justified. Which finds application in many of its various fields of study, from the calculation of naval structures to calculate the hydrostatic stability of the ship, through the alignment and smoothing the hull form.

The project will begin by stating some basic concepts of numerical methods as well as the number of errors that we can make in these and that we must consider. Next will be explained in detail each of the methods we have considered divided into different themes according to their application, starting with solving nonlinear equations, continuing with solving linear systems, numerical interpolation, differentiation and numerical integration, ending with the study of methods of approximation of differential equations of the first order.

As a final project of a Degree in naval engineering and marine systems architecture, each chapter will attempt to solve a case using one of the methods explained from the corresponding chapter on the naval design.

Índice de Contenidos

Índice de figuras.

Capítulo I.

Capítulo II.

Capítulo III.

Capítulo IV.

Capitulo V.

Capítulo VI.

Figura 6.27. Representación gráfica de los resultados obtenidos en el método de Runge-Kutta para sistemas sin solución exacta en la variable y.

Objetivos.

En este proyecto nos planteamos el objetivo de conocer en profundidad, tanto matemáticamente como su implantación informática, herramientas de métodos numéricos que se utilizan en el campo de la hidrostática y/o el ámbito naval.

Serán elaboradas, para cada uno de los métodos que serán descritos, interfaces gráficas de usuario en el programa Matlab con el fin de simplificar el uso de programas informáticos creados al efecto de resolver ciertos métodos numéricos que aparecen normalmente en aplicaciones relacionadas con el ámbito naval.

Para ello seguiremos o realizaremos las siguientes fases:

1. Estudio de los métodos numéricos necesarios para resolver ciertos problemas en el ámbito naval. Este campo de aplicación será muy variado y podrá incluso extenderse a la solución de problemas de otros ámbitos de la industria.

2. Implementación de programas informáticos para su resolución. Para ello utilizaremos el lenguaje de programación Matlab.

3. Preparación de una memoria. En ella quedará reflejado el trabajo realizado ejemplificando algunos de los métodos con ejemplos prácticos relacionados con el ámbito naval y que hayan sido resueltos con los programas anteriormente desarrollados.

4. Elaboración y lectura de una presentación adecuada del proyecto.

En última estancia, este proyecto persigue a su vez un fin docente, permitiendo al alumno, través de éste proyecto y de sus interfaces gráficas para cada uno de los métodos, más visuales y tangibles, alcanzar la comprensión de los pasos seguidos en los distintos métodos iterativos de cálculo y cada uno de sus algoritmos.

De ésta manera, el futuro estudiante dispondrá de una herramienta más para llevar a cabo con éxito el estudio de una materia tan compleja como fascinante y necesaria para cualquier alumno de ingeniería o ciencias como son los métodos de cálculo numérico.

Debemos señalar que dichos algoritmos están presentes y son la base de programas más avanzados de diseño ingenieril, utilizados diariamente por profesionales del sector, para la resolución y aproximación de soluciones que serán tomadas como ciertas, a pesar del posible error cometido, transformándose en decisiones críticas de cualquier proceso constructivo o de diseño.

Capítulo 1. Conceptos Básicos.

Concepto de método iterativo.

Un método iterativo será aquel que, partiendo de una solución aproximada, permite la obtención de un resultado que aproxima de alguna manera a la solución del problema en cuestión.

Para ello, dicho método seguirá un conjunto de reglas u algoritmo que le permitirá obtener mediante un número finito de operaciones elementales, dichos resultados aproximados, cometiendo, en general, un error que deberá ser igual o menor al considerado como aceptable.

Un algoritmo es un procedimiento que describe, sin ambigüedad posible, una sucesión finita de pasos que hay que realizar en un orden preciso, desde la introducción de datos hasta la obtención de resultados.

$$Entrada \rightarrow Proceso \rightarrow Salida$$

Como vehículo para describir los algoritmos se utilizará un pseudocódigo que especifica tanto los datos de entrada, como el proceso que se debe realizar sobre los mismos para la obtención de resultados deseados, así como la forma de salida de éstos últimos.

Concepto de convergencia

Para cada método debemos ser capaces de asegurar de alguna manera que la sucesión de soluciones obtenidas $x_n$es cada vez más próxima a la solución real que queremos obtener x.

El método numérico correspondiente se dice que es convergente si es capaz de proporcionar una solución aproximada del problema considerado para cualquier grado prefijado de exactitud, es decir, si x_n es la solución obtenida en n pasos y s es la solución exacta dado $\varepsilon > 0$, existe $n_0 \in N$ tal que

$$|x_{n_0} - s| < \varepsilon$$

Concepto de estabilidad

Otro requisito de los métodos numéricos es el de que sean estables, que significa, que pequeñas modificaciones en los datos no ocasionen fuertes cambios en el resultado final; o de otra forma si los cálculos no se efectúan con exactitud (debido a los redondeos) no obstante se tiene convergencia a la solución.

El algoritmo será estable si a perturbaciones pequeñas a la entrada, dan perturbaciones pequeñas a la salida.

$d(f_s, \tilde{f}_s) \leq \sigma\left(d(f_0, \tilde{f}_0)\right) \rightarrow lim_{x \rightarrow 0}\sigma(x) = 0$ Algoritmo estable.

Es decir, la diferencia entre los resultados de salida sin modificar f_s y los modificados $\tilde{f}_s$ debe ser menor o igual a la diferencia entre los datos de entrada sin modificar f_0 y los modificados $\tilde{f}_0$ multiplicados por un coeficiente σ siendo el valor de éste coeficiente igual a cero cuando x tiende a cero.

Por otro lado, es necesario demostrar la existencia de algoritmos inestables cuyos resultados quedan variados severamente debidos a pequeñas variaciones en los datos de entrada. Para reflejar este concepto desarrollamos el siguiente ejemplo;

Ejemplo de algoritmo inestable.

$$I_n = \int_0^1 \frac{x^n}{a+x} dx = \int_0^1 \frac{x * x^{n-1}}{a+x} d$$

$$= \int_0^1 \frac{(a+x-a) * x^{n-1}}{a+x} dx = \int_0^1 x^{n-1}\, dx - \int_0^1 \frac{a}{a+x} x^{n-1} dx = \left[\frac{x^n}{n}\right]_0^1 - a *= \frac{1}{n} - a * I_{n-1}$$

$I_0 inicial$

$$I_n = \frac{1}{n} - aI_{n-1}$$

$$I_0 = \int_0^1 \frac{1}{a+x} dx = Ln[|a+x|]_0^1 = Ln|\frac{a+1}{a}|$$

Probamos numéricamente, asignamos valores a=11 con 10 iteraciones:

$$I_0 = Ln\frac{12}{11} = 0{,}087011376$$

$$I_1 = \frac{1}{1} - 11 * I_0 = 1 - 11 * 0{,}087011376 = 0{,}042874853$$

$$I_2 = \frac{1}{2} - 11 * I_1 = 0{,}028376615$$

……..

$$I_{10} = \frac{1}{10} - 11 * I_9 = 0{,}007609187$$

Si implantamos éste algoritmo en Matlab, obtenemos el siguiente resultado.

im = -1.0407e+008 Valor de la integral calculada de forma recursiva.

Por lo cual el error es muy distinto y no se considera aceptable.

Si llamamos $I^*{}_0 = I_0 + \varepsilon_0$, donde ε_0 es el error,

$$I_n - I^*{}_n = \frac{1}{n} - a * I_{n-1} - \left(\frac{1}{n} - aI^*{}_{n-1}\right) = -a * (I_{n-1} - I^*{}_{n-1})$$

El error cometido en cada iteración del método recursivo será;

$\varepsilon_n = I_n - I^*{}_n$

$\varepsilon_n = (-a) * \varepsilon_{n-1} = (-a)^2 * \varepsilon_{n-2} = (-a)^n * \varepsilon_0$

Los algoritmos de éste tipo con crecimiento exponencial del error deben ser desertados del cálculo numérico.

Análisis del error.

En la práctica del cálculo numérico es importante tener en cuenta que las soluciones calculadas por el computador no son soluciones matemáticas exactas. La precisión de una solución numérica puede verse disminuida por diversos factores, algunos de naturaleza sutil, y la compresión de éstas dificultades puede guiarnos a menudo a desarrollar o construir algoritmos numéricos adecuados.

- ### Error absoluto y error relativo.

Supongamos que $\hat{p}$ es una aproximación a p. El error absoluto de la aproximación es $E_p = |p - \hat{p}|$ y el error relativo es $R_p = |p - \hat{p}|/|p|$, supuesto que $p \neq 0$.

El error absoluto no es más que la distancia entre el valor exacto y el valor aproximado, mientras que el error relativo mide el error entendido como una porción del valor exacto.

Diremos que un número $\hat{p}$ es una aproximación a p con d cifras decimales significativas si d es el mayor número natural tal que

$$\frac{|p - \hat{p}|}{|p|} < \frac{10^{-d}}{2}$$

- ### Orden y rapidez de convergencia: notaciones o y O.

Las notaciones o y O de Landau para funciones.

Sean f y g aplicaciones reales definidas en un entorno del punto x_0,decimos que f es una o minúscula de g cuando x tiende a x_0 (y escribimos $f(x) = o\big(g(x)\big)\ cuando\ x \to x_0$) si para x próximo a x_0 con $x \neq x_0$,g no se anula y $lim_{x \to x_0}\left(\frac{f(x)}{g(x)}\right) = 0$,es decir si para todo ε>0 existe un r>0, tal que para todo x verificando $0 < |x - x_0| < r\ es\ |f(x)| < \varepsilon|g(x)|$; la definición viene a expresar la idea de que f(x) se vuelve despreciable frente a g(x) cuando x tiende a x_0.

Otra notación habitual para hablar del tamaño relativo de las funciones en las proximidades de un punto es la siguiente: decimos que f es una O mayúscula de g cuando x tiende a x_0 (y escribimos $f(x) = O\big(g(x)\big) cuando\ x \to x_0$) si existen constantes K y r positivas tales que para todo x verificando $|x - x_0| < r\ es\ |f(x)| \leq K|g(x)|$.

En ocasiones lo que interesa es comparar una función f(x) con monomios $g(x) = (x - x_0)^m$ cuando $x \to x_0$, así hablamos de que $f(x) = o((x - x_0)^m)$ cuando $x \to x_0$ para expresar que f(x) es un infinitésimo de orden superior a $(x - x_0)^m$ cuando $x \to x_0$, es decir f(x) converge a cero más rápidamente que lo hace $(x - x_0)^m$ cuando $x \to x_0$.

Ejemplo.

1. ¿Es $\cot x = o(x^{-1})$ cuando $x \to 0$?

$$lim_{x\to 0} \frac{\cot x}{1/x} = lim_{x\to 0} x cot x = lim_{x\to 0} \frac{\cos x}{\sin x} = 1$$

Por tanto no se verifica que cot x sea una $o(x^{-1})$

2. Consideremos las funciones $f(x) = x^3 + 2x^2$ y $g(x) = x^2$. Puesto que $x^3 \leq x^2$ para $|x| \leq 1$, obtenemos que $x^3 + 2x^2 \leq x^2$ para $|x| \leq 1$. Por tanto $f(x) = O(g(x))$

Orden de aproximación $O(h^n)$ para una sucesión.

Está claro que las sucesiones $\{\frac{1}{n^2}\}_{n=1}^{\infty}$ y $\{\frac{1}{n}\}_{n=1}^{\infty}$ son ambas convergentes a cero pero, sin embargo, debemos hacer notar que la primera sucesión converge a cero más rápidamente que la segunda.

Sean $\{x_n\}_{n=1}^{\infty}$ e $\{y_n\}_{n=1}^{\infty}$ dos sucesiones. Se dice que la sucesión $\{x_n\} es\ de\ orden\ \{y_n\}$, lo que denotamos por $x_n = O(y_n)$, si existe constantes C y N tales que

$$|x_n| \leq C|y_n| \text{ siempre que n} \geq \text{N}$$

Ejemplo

$\frac{n^2-1}{n^3} = O(\frac{1}{n})$, ya que $\frac{n^2-1}{n^3} \leq \frac{n^2}{n^3} = (\frac{1}{n})$ simpre que n≥1.

Sin embargo

$\frac{n+1}{n^3} = o(\frac{1}{n})$ cuando $n \to \infty$ ya que $lim_{n\to\infty} \frac{\frac{n+1}{n^3}}{\frac{1}{n}} = lim_{n\to\infty} \frac{n^2+n}{n^3} = 0$.

Dada una sucesión de números reales $\{x_n\}$ converge a un número real x, se dice que la convergencia es:

- Lineal si existe una constante positiva $c < 1$ y un entero n_0 tal que $|x_{n+1} - x| \leq c|x_n - x| para\ todo\ n \geq n_0$.
- Superlineal si existe una sucesión convergente a cero ξ_n y un número entero n_0 tal que $|x_{n+1} - x| \leq \xi_n|x_n - x| para\ todo\ n \leq n_0$.

- De orden, al menos ,$p > 1$ si existen dos constantes K(no necesariamente menor que 1) y n_0 tal que $|x_{n+1} - x| \leq K|x_n - x|^p$ para todo $n \geq n_0$. Si $p = 2$, la convergencia se dice, al menos, cuadrática; si $p = 3$ cúbica, etc.

Tipos de errores.

1. Error en los datos iniciales

Por ejemplo si son resultado de la medida con algún instrumento, el cual posee un error de medida dependiente de su tolerancia.

2. Error de redondeo.

La representación de los números reales en un computador está limitada por el número de cifras de la mantisa, de manera que algunos números no coinciden exactamente con su representación en el ordenador. Esto es lo que se conoce como error de redondeo. El número que, de hecho, se guarda en la memoria del computador puede haber sufrido la poda o el redondeo de su última cifra, en consecuencia, y puesto que el computador trabaja con números que tienen una cantidad limitada de cifras, los errores de redondeo se introducen y propagan cuando se hacen varias operaciones sucesivas.

- Error de la división por 10.

Cuando representamos una fracción en forma binaria, lo usual es que esta representación sea periódica; por ejemplo,

$$\frac{1}{10} = 0.0\overline{0011}_{dos}$$

Cuando usamos una mantisa con 32 cifras, el computador trunca y usa como aproximación interna

$$\frac{1}{10} \approx 0.11001100110011001100110011001100_{dos}x2^{-3}$$

Cuyo error, diferencia entre las dos ecuaciones, es

$$0.\overline{1100}_{dos}x2^{-35} \approx 2.328306437x10^{-11}$$

Por este motivo, el computador tiene que cometer un error cuando le pedimos que sume 1/10 mil veces. Este error en la suma debería ser al menos

$$(100000)(2.328306437x10^{-11}) = 2.328306437x10^{-6}$$

El error es, de hecho, mucho mayor ya que habrá ocasiones en que sea necesario redondear la suma parcial, además, conforme la suma crece, los sumandos 1/10 son pequeños comparados

con el tamaño que tenga la suma en ese momento por lo que su contribución se trunca de forma más severa. El efecto total de estos errores produce un error final que vale

$$10\,000 - 9999.99447 = 5.53x10^{-3}$$

- Epsilon máquina.

Definimos el épsilon de máquina ϵ como el número positivo más pequeño que al sumar a otro número a verifica que $\epsilon + a > a$, de manera que todo número positivo ϵ' menor que ϵ verifica que $\epsilon' + a = a$. Nos indica el orden de exactitud de la máquina.

Implementamos un algoritmo en Matlab, para calcular el orden de exactitud del ordenador;

El épsilon de la máquina es

2.2204e-016

- Suma de la serie de inversos cuadrados.

Si realizamos la suma iterativa de una serie de coeficientes de 10 elevados sucesivamente un orden superior hasta un orden n, vemos como al ser cada vez el sumando a añadir de orden menor vemos una pérdida de cifras significativas, cometiendo un error en el resultado.

Sin embargo hayamos una clara diferencia al realizar la operación de forma progresiva o regresiva. Al realizar la operación de forma progresiva, es decir, empezando a sumar por el coeficiente de orden mayor, la pérdida de cifras significativas es mucho mayor. Esto se debe a que el nuevo sumando en cada iteración es de orden menor, y así cada vez menor, que el sumando anterior. Sin embargo, al realizar la operación de forma regresiva, el error cometido es considerablemente menor.

El algoritmo a implementar será el siguiente;

$$\sum_{n=1}^{\infty} \left(\frac{1}{10}\right)^n$$

3. Error de truncamiento ó discretización.

Se llama discretización de un problema, dado por un proceso infinito, a la sustitución del mismo por la solución de un problema con un número finito de pasos.

Por ejemplo, el método numérico dado por la inducción,

$$P_N \begin{cases} x_0 = f(a) \\ x_n = x_{n-1} + \dfrac{f^{(n)}(a)}{n!} - (x-a)^n, 1 \leq n \leq N, \end{cases}$$

Es una discretización del cálculo de $f(x)$ mediante el desarrollo limitado de Taylor de orden N de f centrado en a.

Por lo tánto en este cálculo será omitido el siguiente término del desarrollo de Taylor;

$$\left|R_{n_0}(f,x)\right| = |f(x) - \sum_{n=0}^{n_0} \frac{f^{n_0+1}(a)}{n!}(x-a)^n| < \varepsilon$$

Por lo que cometeremos un error.

Para aplicar el truncamiento a una función, dos de los métodos mas utilizados son las series de Taylor y Fourier, que describimos brevemente a continuación;

- Desarrollo de Taylor y MacLaurin

Supongamos que $f \in C^{n+1}[a,b]$ y $x_0 \in [a,b]$. Entonces, para cada $x \in (a,b)$ existe un número $c = c(x)$ (el valor de c depende de x) que está entre x_0 y x y verifica

$$f(x) = P_n(x) + R_n(x)$$

Donde

$$P_n(x) = \sum_{k=0}^{n} \frac{f^{(k)}(c)}{k!}(x-x_0)^k$$

Se llama polinomio de Taylor de grado n de f alrededor de x_0 y

$$R_n(x) = \frac{f^{(n+1)}(c)}{(n+1)!}(x-x_0)^{n+1}$$

En particular si c=0 el desarrollo anterior se denomina de MacLaurin y queda como sigue

$$f(x) = \sum_{k=0}^{n} \frac{f^k(0)}{k!}x^k + \frac{f^{(n+1)}(\theta x)}{(n+1)!}x^{n+1}$$

Con $\theta \in (0,1)$

- Series de Fourier.

Toda función $f(t)$ periódica de periodo 2P, se puede representar en forma de una suma infinita de funciones armónicas, es decir,

$$f(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} \left(a_k cos \frac{k\pi t}{P} + b_k sin \frac{k\pi t}{P} \right)$$

Donde $a_0, a_1 \dots a_k \dots y\ b_0, b_1, \dots b_k$ son los denominados coeficientes de Fourier.

Los coeficientes del desarrollo en serie valen

$$a_k = \frac{1}{P} \int_{-P}^{P} f(t) cos \frac{k\pi t}{P} dt$$

$$b_k = \frac{1}{P} \int_{-P}^{P} f(t) sin \frac{k\pi t}{P} dt$$

La suma parcial de las series de Fourier es

$$s_n(t) = \frac{a_0}{2} + \sum_{k=1}^{n} \left(a_k cos \frac{k\pi t}{P} + b_k sin \frac{k\pi t}{P} \right)$$

Si la función $f(t)$ tiene simetría, algunos de los coeficientes resultan nulos.

- Si $f(t)$ es una función par, $f(t) = f(-t)$, los términos b_k son nulos .
- Si $f(t)$ es una función impar $f(t) = -f(-t)$, los coeficientes a_k son nulos.

Para ejemplificar el desarrollo de Fourier implementamos dicho algoritmo en Matlab, que sea capaz de mostrar, en los intervalos deseados, la función que deseamos y compararla con la serie de Fourier obtenida.

- Error discretización primera derivada.

Al realizar el cálculo de la derivada de una función cometemos un error al despreciar o truncar una serie de términos. A su vez, dicho error variará si el cálculo lo realizamos de forma progresiva, regresiva o centrada, siendo ésta ultima la más aconsejable como demostraremos a continuación.

Si realizamos el cálculo de la primera derivada de una función de **forma progresiva** entorno a dos puntos (c) y su próximo (c+h)

El desarrollo de la serie de Taylor de f(c+h) será;

$$f(c+h) = f(c) + f'(c)h + \frac{f''(c)}{2!}h^2 + \cdots$$

Que restando f(c) y dividiendo por h obtenemos la expresión de la derivada;

$$\frac{f(c+h) - f(c)}{h} = f'(c) + \frac{f''(c)}{2!}h + \cdots = f'(c) + O(h)$$

Si, a continuación, realizamos el cálculo de la primera derivada de una función de **forma regresiva** entorno a dos puntos (c) y su próximo (c-h)

El desarrollo de la serie de Taylor de f(c-h) será;

$$f(c-h) = f(c) - f'(c)h + \frac{f''(c)}{2!}h^2 - \frac{f'''(c)}{3!}h^3 + \cdots$$

Que restando f(c) y dividiendo por h obtenemos la expresión de la derivada;

$$\frac{f(c) - f(c-h)}{h} = f'(c) - \frac{f''(c)}{2!}h + \cdots = f'(c) + O(h)$$

Como hemos explicado anteriormente, con el término $O(h)$ expresamos que la serie es de convergencia lineal o de primer orden y su error disminuye a la misma velocidad que lo hace h.

Por último, si realizamos el cálculo de la primera derivada de una función de **forma centrada** entorno a dos puntos (c+h) y su próximo (c-h).

El desarrollo de la serie de Taylor de f(c+h) y f(c-h) será;

$$f(c+h) = f(c) + f'(c)h + \frac{f''(c)}{2!}h^2 + \frac{f'''(c)}{3!}h^3 + \cdots$$

$$f(c-h) = f(c) - f'(c)h + \frac{f''(c)}{2!}h^2 - \frac{f'''(c)}{3!}h^3 + \cdots$$

Al restar ambas expresiones y dividir por 2h, obtenemos la expresión de la primera derivada;

$$\frac{f(c+h) - f(c-h)}{2h} = f'(c) + \frac{f'''(c)}{3!}h^2 + \cdots = f'(c) + O(h^2)$$

Como hemos explicado anteriormente, con el término $O(h^2)$ expresamos que la serie es de convergencia cuadrática o de segundo orden, es decir, que el error en la aproximación tiende a cero a la misma velocidad que lo hace h^2. Por lo tanto, ésta última serie converge a la solución más rápidamente que las dos anteriores conforme disminuye h.

- Error discretización segunda derivada.

Al realizar el cálculo de la segunda derivada de una función cometemos un error al despreciar o truncar una serie de términos.

Si definimos una función f(x) entorno a tres puntos (c-h), c y (c+h), y desarrollamos las correspondientes series de Taylor de f(x) en torno a (c-h) y (c+h), como ya hemos hecho anteriormente;

$$f(c+h) = f(c) + f'(c)h + \frac{f''(c)}{2!}h^2 + \frac{f'''(c)}{3!}h^3 + \cdots$$

$$f(c-h) = f(c) - f'(c)h + \frac{f''(c)}{2!}h^2 - \frac{f'''(c)}{3!}h^3 + \cdots$$

Al sumar ambas expresiones, obtenemos;

$$f(c+h) + f(c-h) = 2f(c) + f''(c)h^2 + O(h^4)$$

Que al restar ambos términos por $-2f(c)$ y dividir por h^2 , obtenemos la expresión de la segunda derivada;

$$\frac{f(c+h) - 2f(c) + f(c-h)}{h^2} = f''(h) + O(h^2)$$

Por lo tanto, esta serie es de convergencia cuadrática o de segundo orden.

Propagación de errores.

En el caso de que queramos hallar una medida u la cual no es posible medir de forma directa, sino que dependerá de otras variables x,y… relacionadas entre sí por la ecuación

$$u = u(x, y, ..)$$

Se trata aquí de determinar el error de u, es decir, en qué medida afectan a u los errores cometidos en la determinación de las x,y…El error absoluto de u corresponderá al incremento total de la función u.

$$\Delta u = u(x \pm \Delta x, y \pm \Delta y, ..) - u(x, y, ..)$$

Admitiendo que los errores son pequeños en comparación con la medida, se puede utilizar el cálculo diferencial para hallar el error de u. Así pues,

$$du = \frac{\partial u}{\partial x}dx + \frac{\partial u}{\partial y}dy + \cdots$$

Donde $dx, dy, ..$ representan los errores absolutos.

- Problema directo.

Se presenta cuando la relación de dependencia entre las variables u y las variables x,y,…

$$u = u(x, y, ..)$$

Se conocen los valores de x,y,.. y el de sus errores absolutos dx,dy,… El problema consiste en determinar u y su error absoluto du.

$$du = \frac{\partial u}{\partial x} dx + \frac{\partial u}{\partial y} dy + \cdots$$

- Problema inverso.

Dada la relación $u = u(x, y, ..)$ se quiere obtener u pero con una determinada exactitud, es decir, fijando el error máximo que se puede cometer en dicha medida indirecta. Para ello hay que calcular cuánto tienen que valer dx,dy,..para obtener como máximo el du exigido en la medida indirecta, o de otro modo,¿ qué instrumentos se deben emplear en las medidas de las x,y,..? En este caso, pues, se conoce du, y aproximadamente las x,y,.. (es decir, se han tomado medidas pero no se sabe si con el error conveniente para obtener, como límite, el du fijado) y es preciso calcular los adecuados dx,dy,..

A partir de la expresión

$$du = \frac{\partial u}{\partial x} dx + \frac{\partial u}{\partial y} dy + \cdots$$

O lo que es lo mismo

$$du = \sum_{i=1}^{n} |\frac{\delta f_{x_i}}{\delta x_i}(\theta)||\Delta x_i| \ < tol$$

Siendo θ un valor entre la medida tomada y el valor real, pero que por comodidad en la práctica se toma como el valor medido.

Si suponemos que el error absoluto de todas las variables implicadas en la función es el mismo, es decir, todos los instrumentos utilizados tendrán la misma exactitud, podemos despejar dicho parámetro de la función de la siguiente manera;

$$|\Delta x_i| < \frac{tol}{\sum_{i=1}^{n} |\frac{\delta f}{\delta x_i}(\theta)|} \leq \frac{tol}{\sum_{i=1}^{n} min_x |\frac{\delta f}{\delta x_i}(x)|}$$

Los problemas inversos siempre suelen ser más difíciles que los directos, y en particular la solución no es única y para que lo sea deberemos de añadir suposiciones lo más ajustadas a la

realidad posible que nos permita resolver el problema. En este caso hemos supuesto que todos los $|\Delta x_i|$ son iguales.

Evaluación de un polinomio.

Supongamos que escribimos un polinomio P(x) de grado n en la forma

$$P(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0$$

- Método de Horner.

Es una técnica para evaluar polinomios que puede ser vista como una colección de multiplicaciones encajadas. Por ejemplo, un polinomio de quinto grado puede escribirse como una colección de cinco multiplicaciones encajadas

$$P_5(x) = \Big(\Big(\big((a_5 x + a_4)x + a_3\big)x + a_2\Big)x + a_1\Big)x + a_0$$

Sea P(x) un polinomio de grado n y sea x=c un número para el que deseamos evaluar P(c).Definimos;

$b_n=a_n$

$b_k=a_k+cb_{k+1}$ para k=n-1,n-2,….,1,0.

Si definimos

$$Q_0(x) = b_n x^{n-1} + b_{n-1}x^{n-2} + \cdots + b_3 x^2 + b_2 x + b_1$$

Y $R_0=b_0$

$$P(x) = (x - c)Q_0(x) + R_0$$

O sea, $Q_0(x)$ es el polinomio coecifiente de grado n-1 y $R_0=b_0=P(c)$ es el resto de la división de P(x) entre (x-c).

Dato	an	an-1	an-2	…	ak	…	a2	a1	a0
c		cbn	cbn-1	…	cbk+1	…	cb3	cb2	cb1
	bn	bn-1	bn-2	…	bk	…	b2	b1	b0=P(c) Resultado

Por otro lado, los coeficientes del polinomio Q_0 , $b(i)$serán los coeficientes de la primera derivada del polinomio y al realizar la misma operación sobre el polinomio obtenido, obtendremos los coeficientes de la segunda derivada y su resultado como b0. Si continuamos realizando la misma operación sucesivamente obtendremos los coeficientes y el valor del resultado en el punto de las sucesivas derivadas.

Algoritmos inadecuados.

- Ecuación de segundo grado.

Existen ciertos casos en los que al resolver un polinomio cuadrático por el método habitual podemos encontrarnos con que hemos realizado la resta de números muy parecidos, cometiendo un error, debido al orden de exactitud de la máquina, produciéndose una pérdida de cifras significativas en dicho cálculo.

Este error es denominado error catastrófico, que suele tener lugar cuando el valor del coeficiente b es de orden mayor a los restantes a y c. En dicho caso al restar $4ac$ a b^2, $4ac$ es despreciado y a continuación restamos b y b, produciéndose el error catastrófico.

$$ax^2 + bx + c = 0$$

Si utilizamos la fórmula habitual

$$x1 = \frac{-b+\sqrt{b^2-4ac}}{2a} \qquad x2 = \frac{-b-\sqrt{b^2-4ac}}{2a}$$

Esto se puede solucionar de la siguiente manera; multiplicando y dividiendo por el conjugado del numerador:

$$x = \frac{-b \pm \sqrt{b^2-4ac}}{2a} \; \frac{\left(-b \mp \sqrt{b^2-4ac}\right)}{\left(-b \mp \sqrt{b^2-4ac}\right)} = \frac{2}{\left(-b \mp \sqrt{b^2-4ac}\right)}$$

Si $-b > 0$

$$x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a} \qquad x_2 = \frac{2c}{-b+\sqrt{b^2-4ac}}$$

Si $-b < 0$

$$x_1 = \frac{-b-\sqrt{b^2-4ac}}{2a} \qquad x_2 = \frac{2c}{-b-\sqrt{b^2-4ac}}$$

- Resolución de polinomio de grado alto. Ejemplo de Wilkinson.

Otro ejemplo de algoritmo inadecuado es el siguiente. Las raíces de

$$(x-2)^2 = 10^{-6}$$

Son

$$x1 = 2 + 10^{-3} \qquad x2 = 2 - 10^{-3}$$

En cambio las raíces de

$$(x-2)^2 = 0$$

Son

$$x1 = x2 = 0$$

Este ejemplo trivial muestra que un pequeño cambio en un coeficiente de la ecuación polinomial puede dar lugar a un cambio de otro orden en las raíces.

Uno de los ejemplos más interesantes de algoritmo inadecuado fue el estudiado por Wilkinson en 1963. Se trata de calcular las raíces de

$$p(x) = (x-1)(x-2)\dots(x-20) = 0$$

Tras incrementar el coeficiente de x^{19} por 10^{-7}

La ecuación de partida tiene las raíces reales $\{1,2,\dots,20\}$ en tanto que la perturbada $\prod_{i=1}^{20}(x-i) + 10^{-7} * x^{19} = 0$ obtenemos 10 raices reales y 10 complejas conjugadas dos a dos.

Implementamos dicho algoritmo en Matlab, con la posibilidad de variar el coeficiente del polinomio que queramos, añadiendo el valor a dicho coeficiente que consideremos oportuno (épsilon)

Ejemplo de aplicación al ámbito naval.

En nuestro ejemplo aproximaremos una función altamente utiliza en el cálculo hidrostático como es la fórmula de Scribanti para el cálculo del brazo adrizante para grandes ángulos en buques considerados de costados rectos.

$$GZ = \left[\overline{GM} + \overline{BM}\frac{tang(\theta)^2}{2}\right] * sen(\theta)$$

Siendo,

GZ, el brazo adrizante que queremos calcular.

$\overline{GM}$ la altura metacéntrica del buque, medida con anterioridad.

$\overline{BM}$ el radio metacéntrico transversa, obtenido con anterioridad y dependiente de las formas del buque.

θ el ángulo de escora del buque.

Para aplicarlo a un caso práctico suponemos una altura metacéntrica de 0.5 m y un radio metacéntrico de 2m. Por lo tanto nuestra función a desarrollar sería;

$$f(x) = GZ(\theta) = [0.5 + tang(\theta)^2]sen(\theta)$$

Lo haremos para el entorno cercano a cero, con un grado cinco de aproximación y evaluaremos en el punto uno.

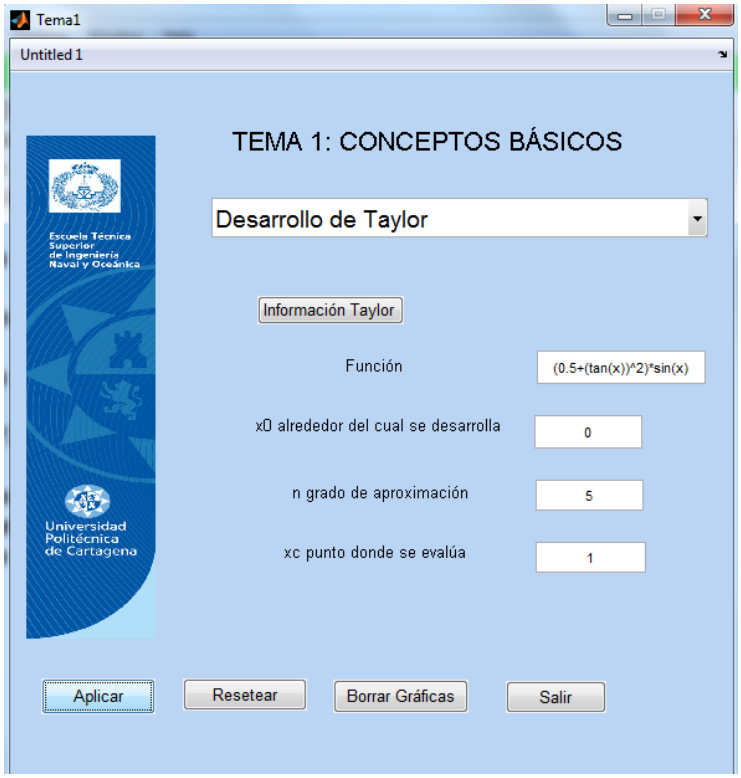


Figura 1.0

Al hacer doble click en el botón aplicar obtenemos los resultados en una ventana emergente;

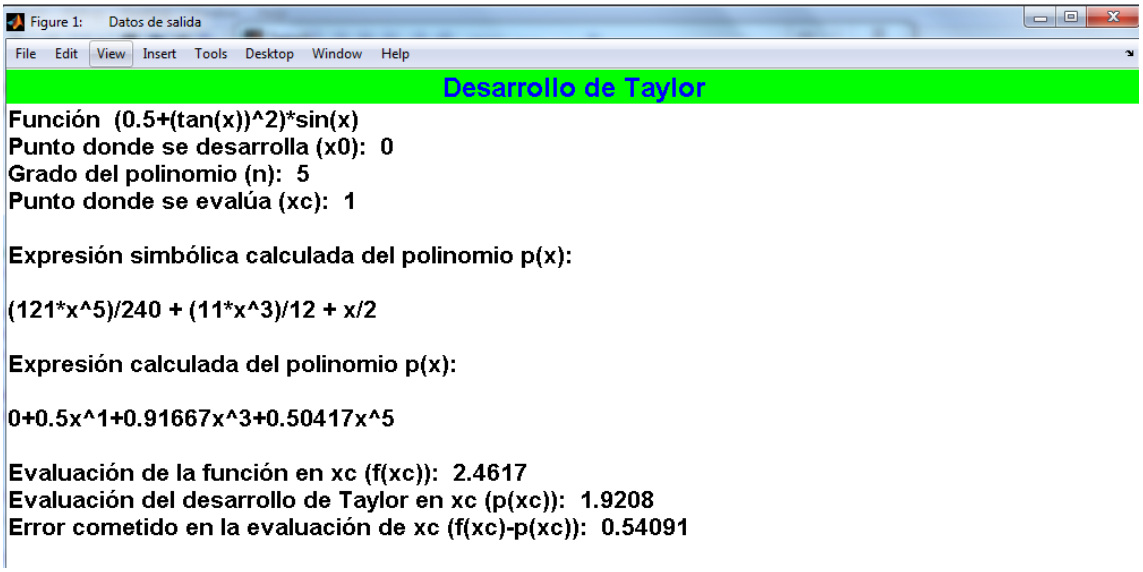


Figura 1.1

Como vemos al aproximar la función por Taylor cometemos un error que consideramos como aceptable y nos permite realizar de forma más cómoda y simple el cálculo de funciones complejas.

Ayuda a la interfaz gráfica.

En éste último apartado desarrollaremos una explicación paso a paso de cómo utilizar la interfaz gráfica que ha sido implementada en el programa de análisis matemático Matlab para una mayor comprensión de la teoría descrita mediante ejemplos prácticos.

Para comenzar debemos iniciar el programa Matlab y situar la carpeta que contiene la interfaz gráfica, así como los demás programas a utilizar en éste capítulo, en la carpeta actual de Matlab denominada "Current folder" y hacer doble click sobre la carpeta "Interfaz gráfica" para que sea ésta a la que el programa recurra cuando realicemos una orden en la ventana de comandos.

Para comenzar introducimos la orden " guide" en la ventana de comandos y en la ventana emergente seleccionamos el fichero de nominado" Tema1_Introducción".

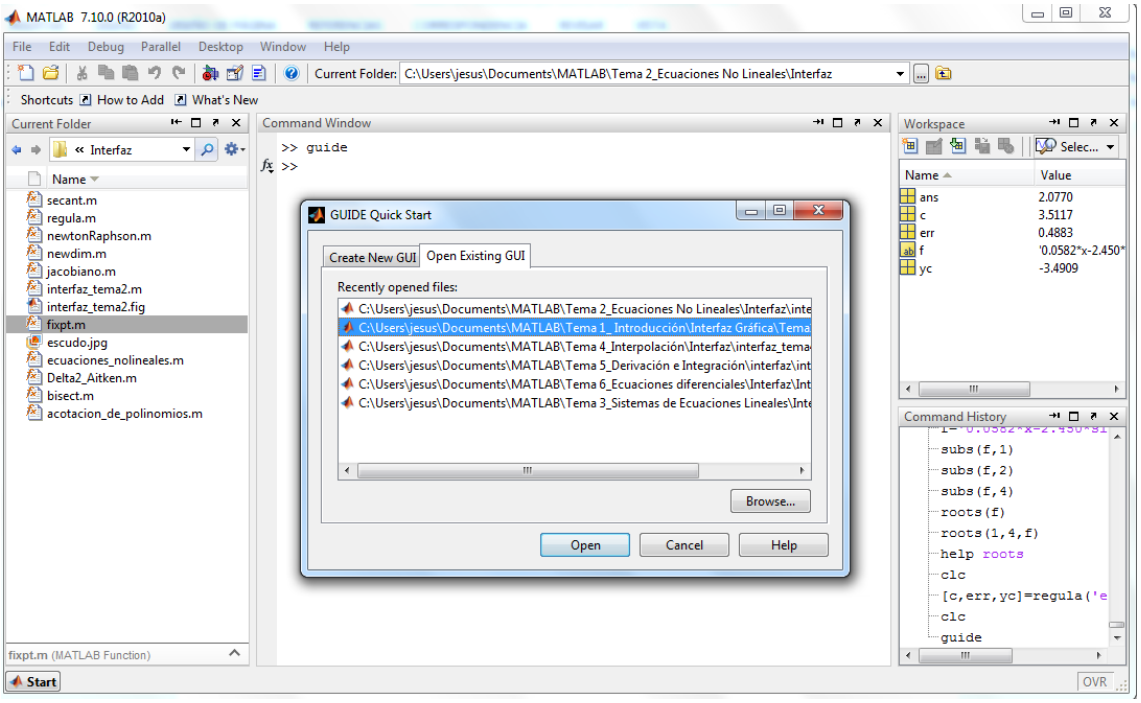


Figura 1.2

Al abrir dicho fichero aparecerá la interfaz de diseño de la propia interfaz gráfica, donde deberemos hacer doble click en el botón verde de reproducir o "play" situado en la zona superior.

Al hacerlo, aparecerá nuestra interfaz gráfica, inicialmente con la lista de métodos que han sido implementados y donde podremos seleccionar el método que queremos reproducir.

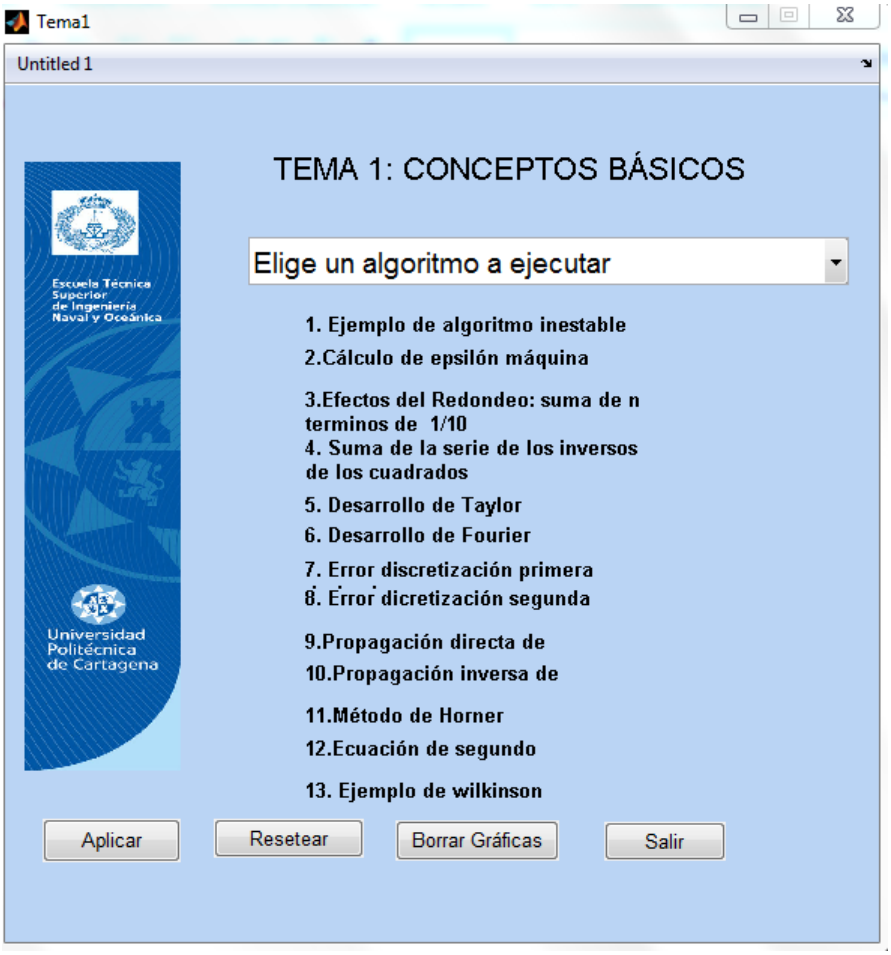


Figura 1.3

1. Ejemplo algoritmo inestable.

Realizamos el ejemplo de la integral inestable antes descrita para valores de a=1 y número de iteraciones n=10. Si llamamos i_b al valor de la integral calculado con el método de integración numérica e i_m al valor de la integral calculado por el método recursivo.

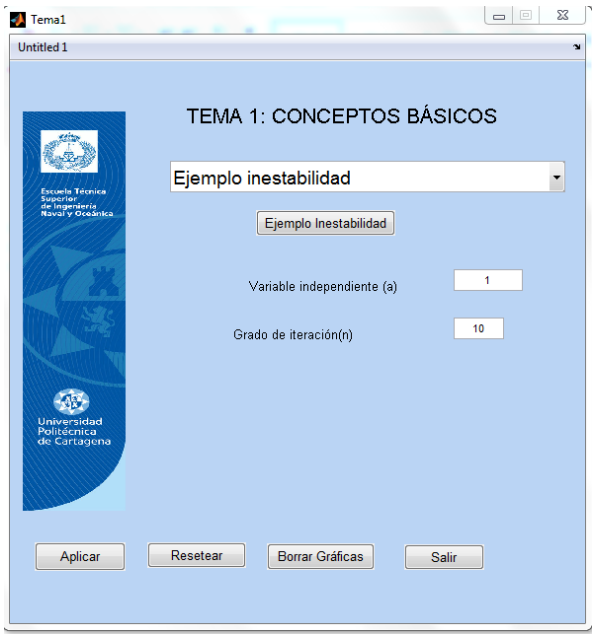


Figura 1.4

A continuación, hacemos doble click sobre en el botón aplicar y obtenemos los siguientes resultados en una ventana emergente.

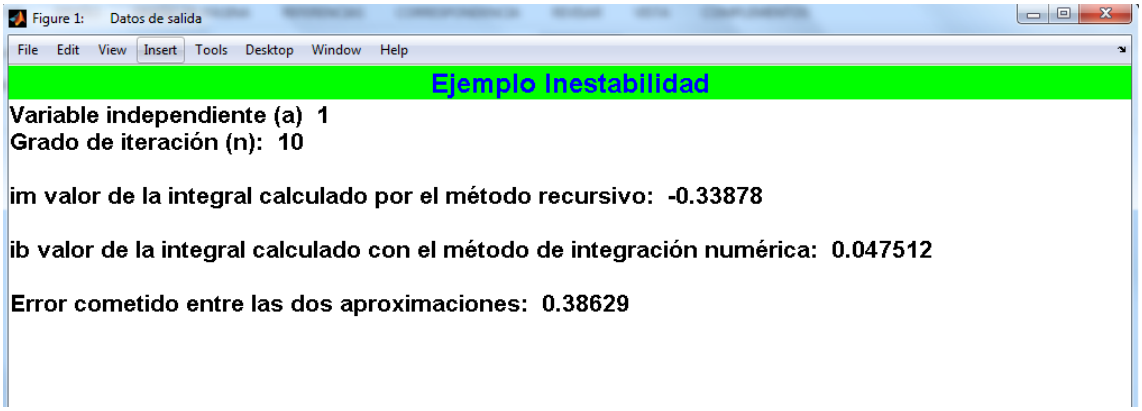


Figura 1.5

Como podemos ver, la diferencia de resultados obtenidos en ambos métodos es considerable incluso para pequeñas variaciones en los datos de entrada, por lo que consideramos el algoritmo como altamente iniestable.

2. Epsilon máquina.

Seleccionamos el método en la ventana principal de la interfaz y, al no exisitir variables de entrada, simplemente hacemos doble click en el el botón aplicar.

Figura 1.6

Nos aparecerá el resultado en la siguiente ventana emergente;

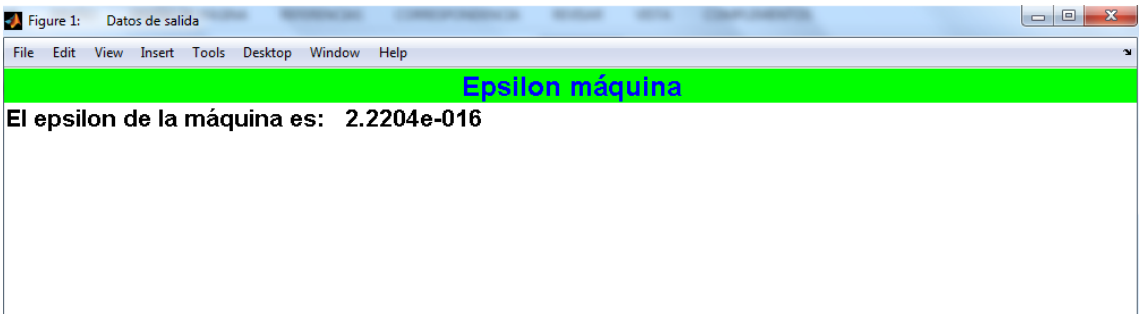


Figura 1.7

Este será el mínimo valor que nuestro ordenador como un valor operacional. Valores inferiores a éste serán despreciados por la máquina.

3. Error de la división por 10.

Seleccionamos el método en la ventana principal y definimos el número de sumandos que queremos en la suma, en este caso, por ejemplo 1000.

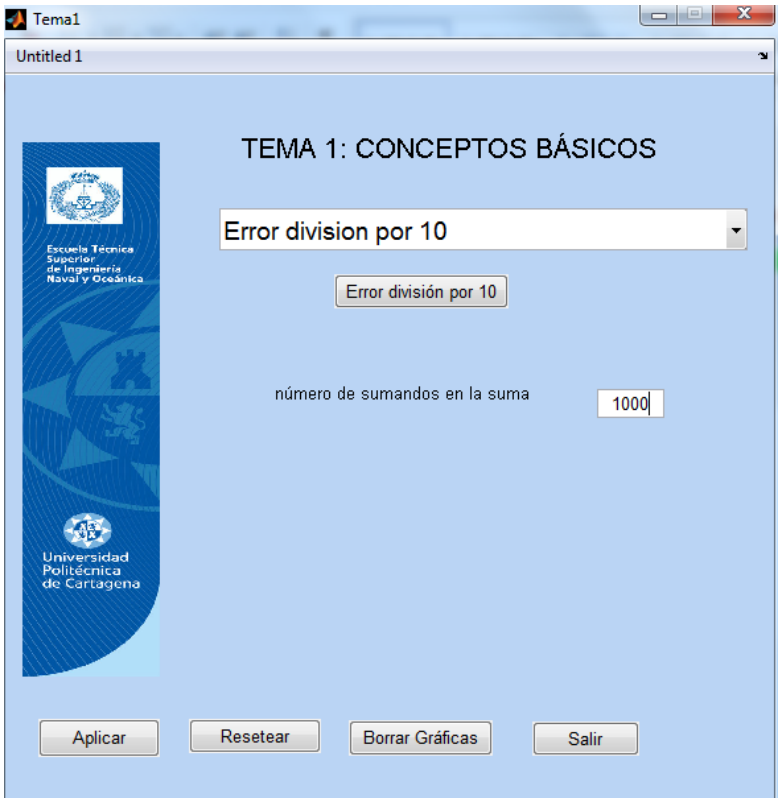


Figura 1.8

Hacemos doble click en el botón aplicar y obtenemos los resultados en la siguiente ventana emergente;

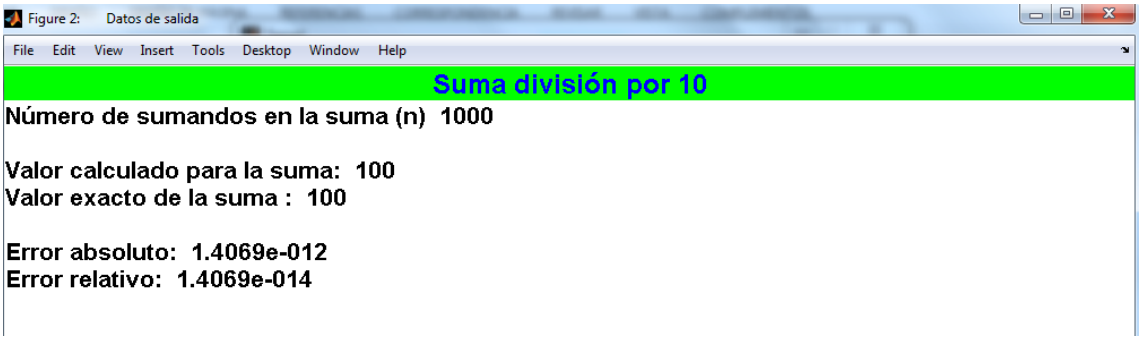


Figura 1.9

Aunque no podamos observarlo a primera vista, el error cometido debido a acciones de redondeo en el cálculo se vuelve considerable.

4. Suma de la serie de inversos cuadrados.

Seleccionamos el método en la ventana principal y elegimos el número de términos que queremos de la serie, en este caso, por ejemplo, 1000 para que pueda observarse la diferencia entre el cálculo de forma progresiva o regresiva.

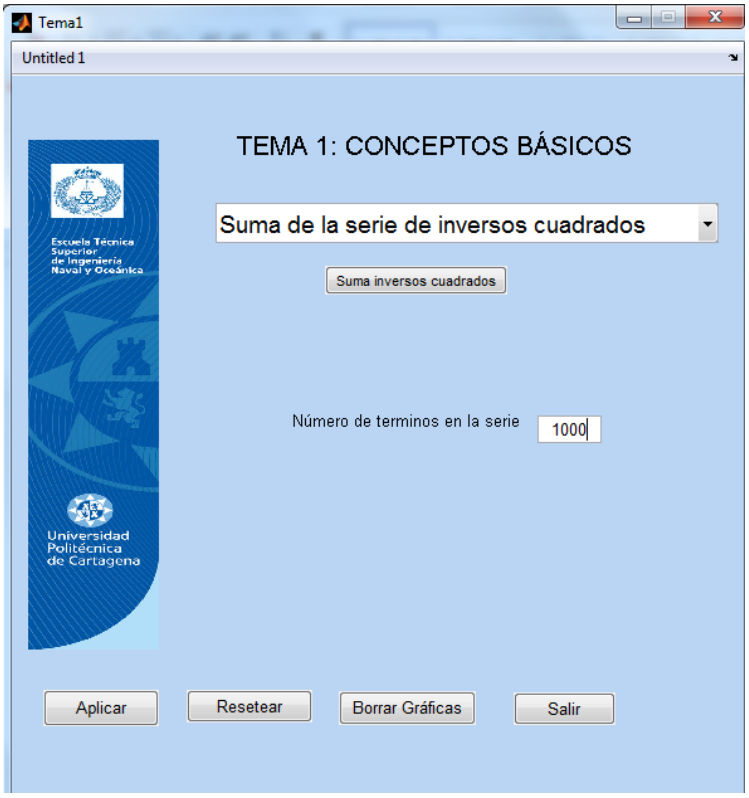


Figura 1.10

Al hacer doble click en el botón aplicar, obtenemos los resultados en una ventana emergente;

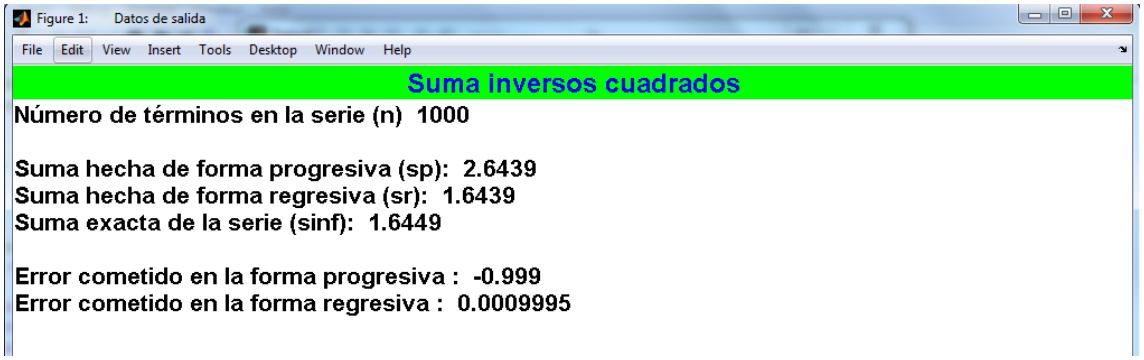


Figura 1.11

Como podemos observar, obtenemos un resultamos mucho más preciso al comenzar sumando por los términos de menor orden hacia los de mayor. En caso contrario cometemos muchos errores al despreciar la máquina los valores de la siguiente iteración o términos de ésta al ser de un orden menor.

5. Desarrollo de Taylor.

Seleccionamos el método en la ventana principal, y nos aparecerán las variables de entrada que debemos introducir.

En este caso tratamos de aproximar la función $f(x) = e^x$ cerca del punto $x_0 = 0$ con un grado de aproximación $n = 4$ en el punto $x = 1$.

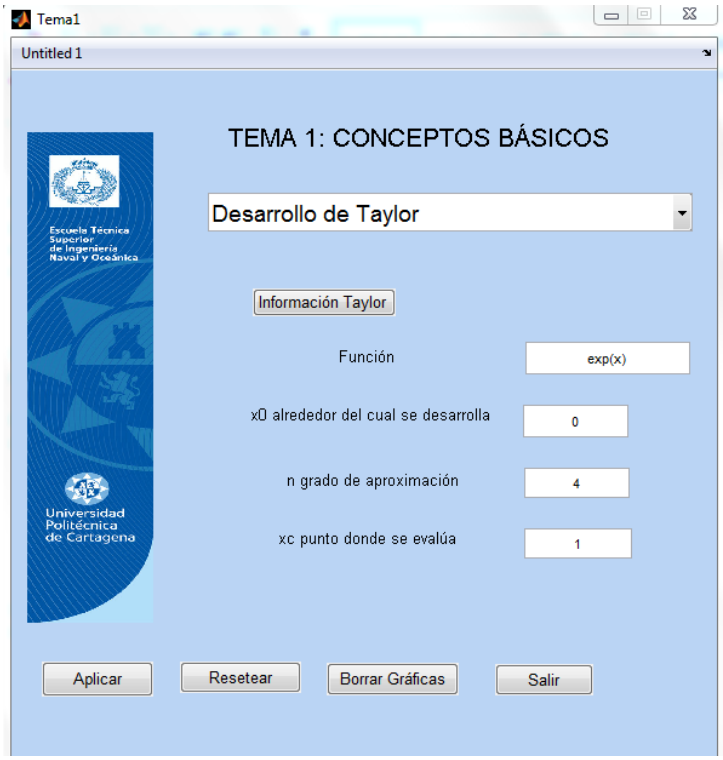


Figura 1.12

Al hace click en el botón aplicar aparece una ventana emergente que contiene los resultados obtenidos,

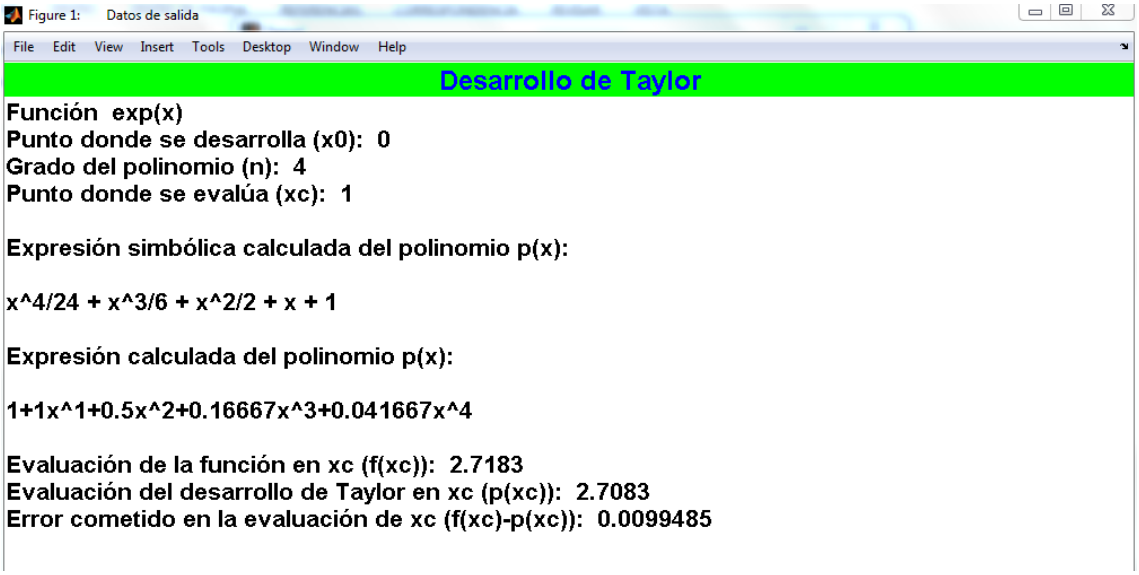


Figura 1.13

Como podemos observar, obtenemos un polinomio que nos permite evaluar, o integrar o derivar la función con mayor facilidad, cometiendo un ligero error de truncamiento al despreciar el resto de términos de la serie de Taylor, pero que consideramos como aceptable, a pesar de no ser el grado de aproximación muy alto.

6. Series de Fourier.

Seleccionamos el método en la ventana principal y nos aparecerán las variables que debemos introducir para llevar a cabo el método.

En primer lugar, si ejemplificamos la función periódica $f(x) = sen(x)$, para tres frecuencias, en un periodo 2pi, un número de frecuencias o grado de aproximación 3 y un número de 2 copias por la izquierda y 3 por la derecha. La función debemos introducirla como una función tipo handle.

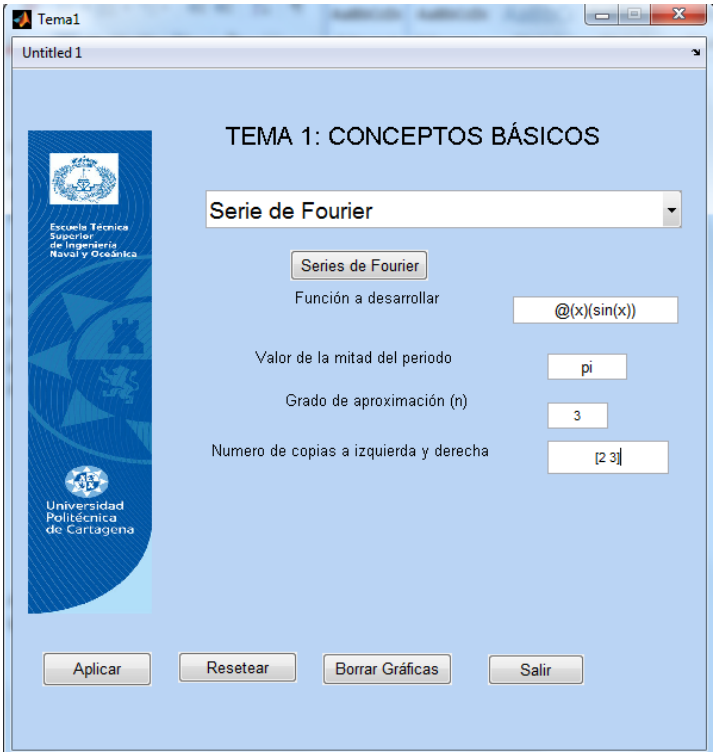


Figura1.14

Al hacer doble click en el botón aplicar obtenemos los resultados en una ventana emergente al igual que una gráfica en la que están representadas ambas funciones.

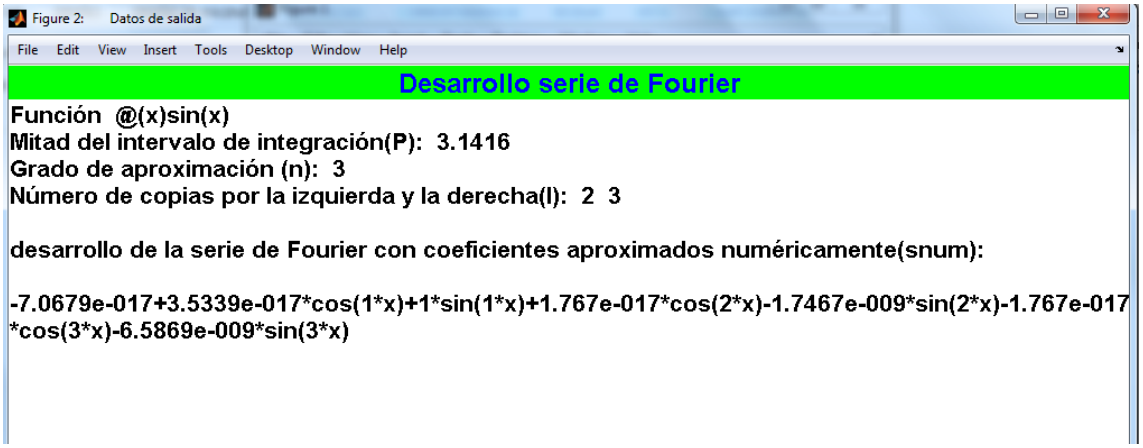


Figura 1.15

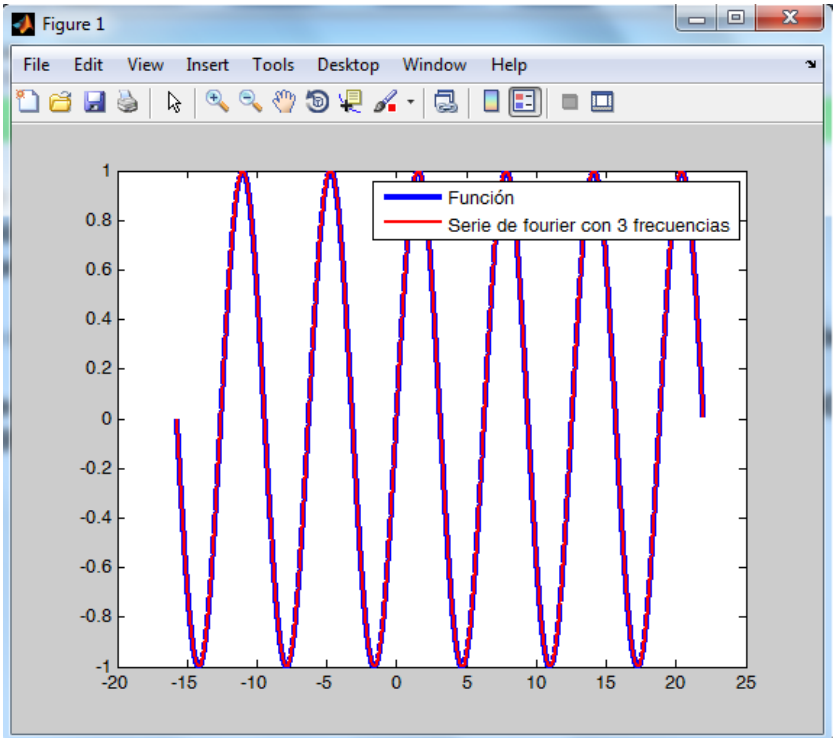


Figura 1.16

Como podemos observar, al tratarse de una función periódica la serie de Fourier calculada se ajusta con gran exactitud a la función.

Sin embargo, si queremos desarrollar la serie de Fourier para una ecuación no periódica como por ejemplo $f(x) = x$ para el mismo número de frecuencias (n=3).

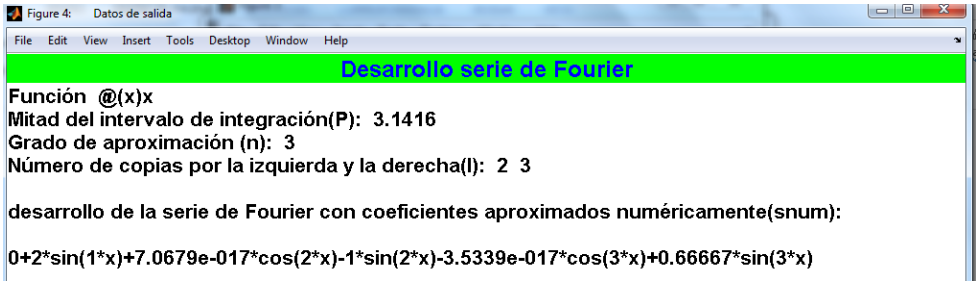


Figura 1.17

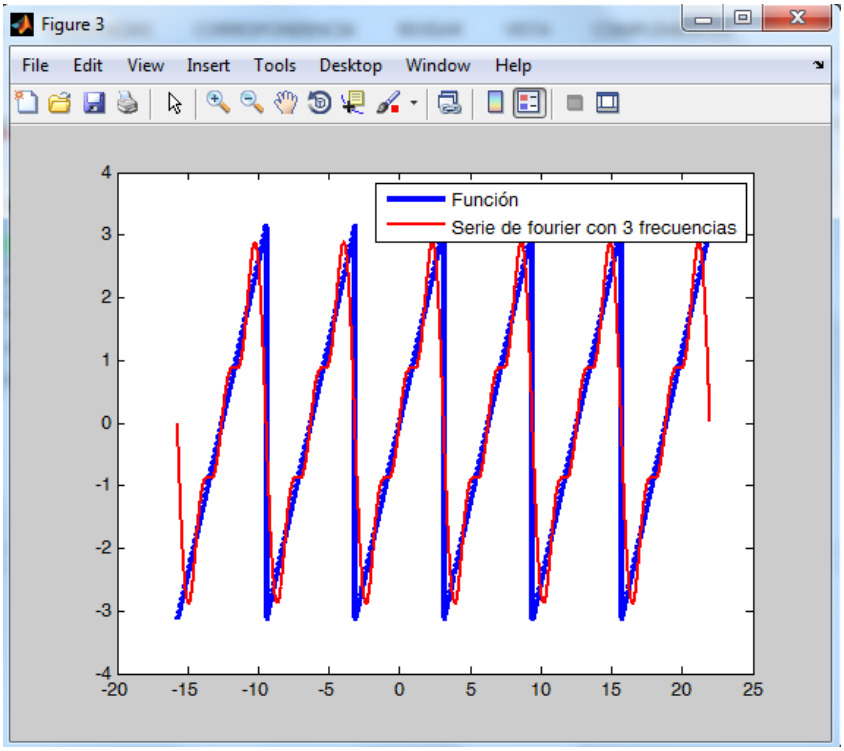


Figura 1.18

Como podemos observar, en los extremos de los intervalos se incrementa el error de aproximación, es el denominado **Efecto de Gips**, según el cual, cuando aproximamos una función no periódica mediante una serie de Fourier, es decir, la forzamos a ser periódica, aparecen unas discontinuidades en los extremos de los intervalos que a mayor número de frecuencias más se propaga el error hacia el error del intervalo.

Conforme aumentamos el grado de frecuencia, podemos observar que más se aproxima la serie a la función, pero, por otro lado, más notable se hace el efecto descrito en los extremos de los intervalos.

Por ejemplo para un número de cinco frecuencias dicho efecto en los extremos se incrementa.

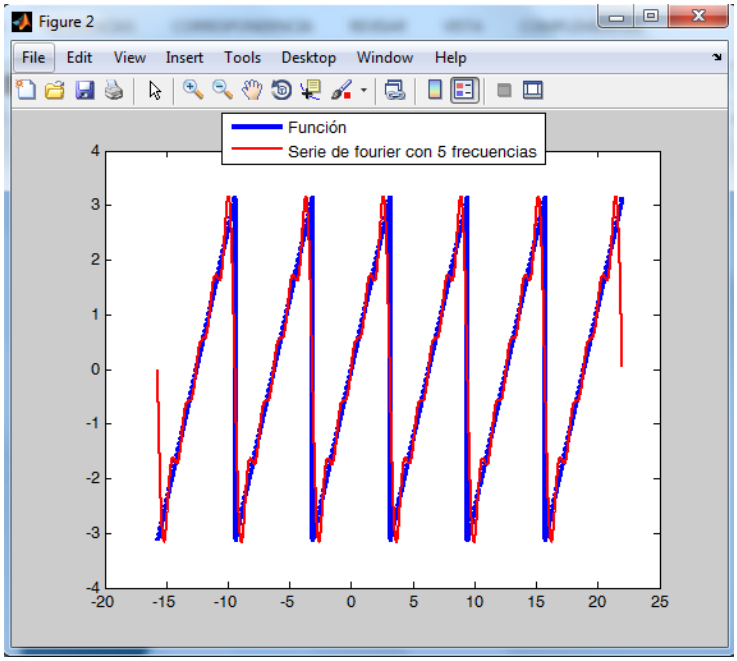


Figura 1.19

Podemos observar cómo se incrementa el llamado efecto de Gips, empeorando la aproximación en los extremos de los intervalos.

7. Error discretización primera derivada.

Seleccionamos el método en la pantalla principal de la interfaz, y nos aparecerán las variables de entrada a introducir.

En este caso realizaremos el ejemplo para la función e^x en el punto de aproximación 1 y un paso de discretización h=1/100.

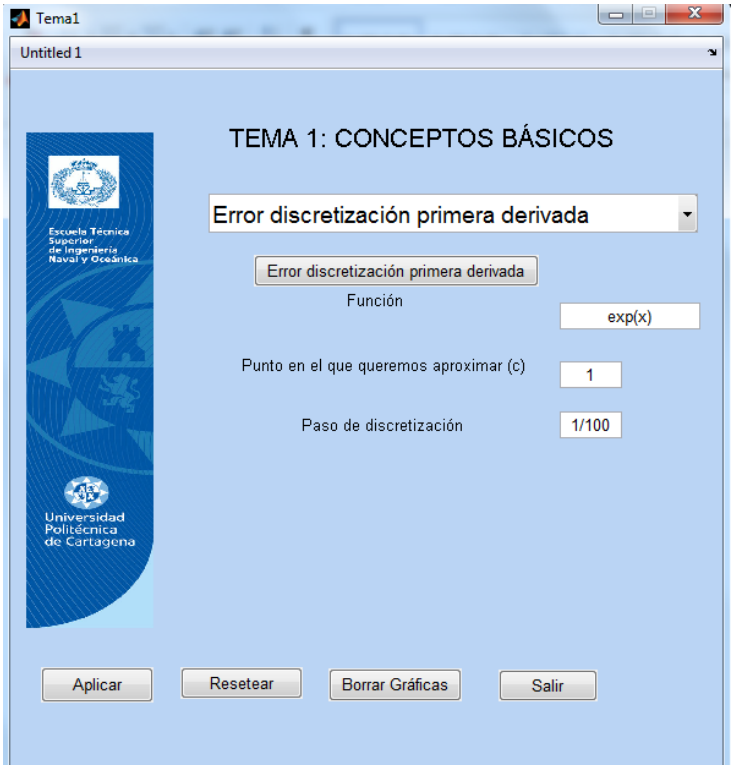


Figura 1.20

Al hacer doble click en el botón aplicar obtenemos los resultados en una ventana emergente;

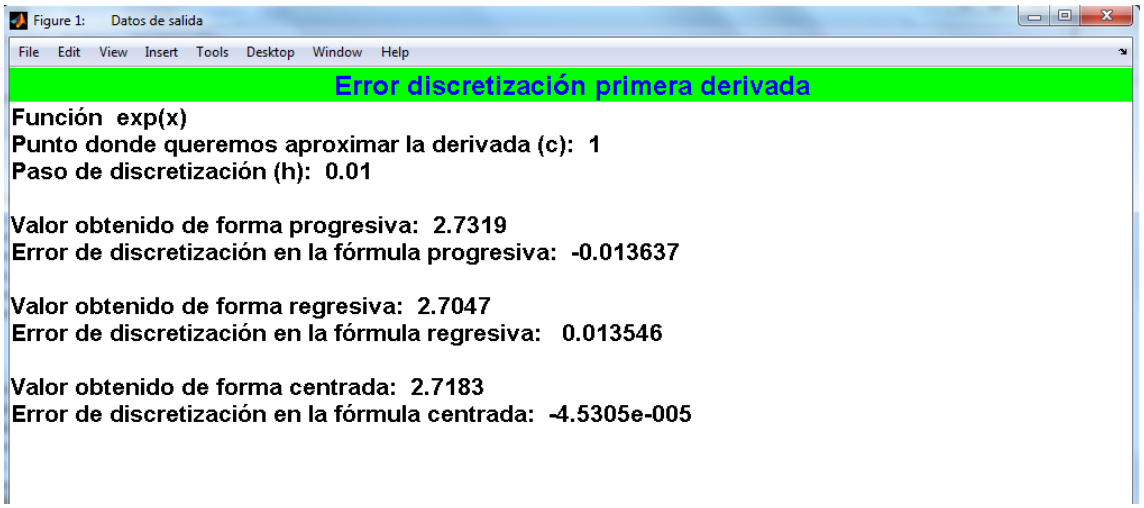


Figura 1.21

Como podemos observar, el error cometido en la forma centrada es la mitad que el cometido en la forma progresiva o regresiva.

8. Error discretización segunda derivada.

Seleccionamos el método en la ventana principal y nos aparecerán las variables que debemos introducir para llevar a cabo el método.

En este caso realizamos un ejemplo con la función, $f(x) = e^{x^2}$ en torno al punto c=1 y paso de discretización h=1/10.

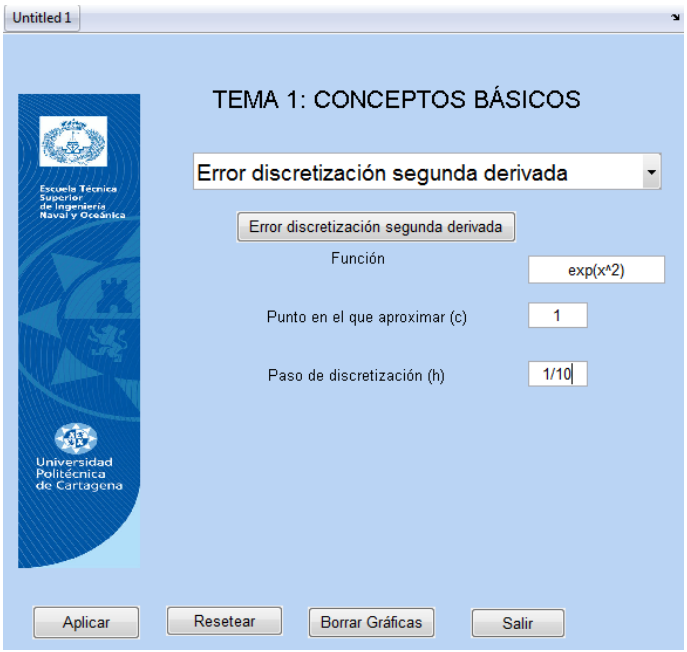


Figura 1.22

Al hacer doble click en el botón aplicar aparece una ventana emergente con los resultados obtenidos.

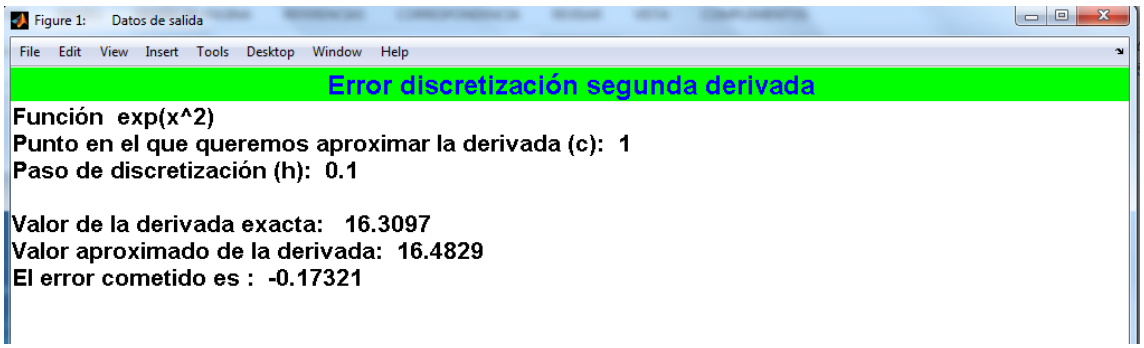


Figura 1.23

Si a continuación reducimos el paso de discretización a la mitad, siendo $h = 1/20$, obtenemos los siguientes resultados;

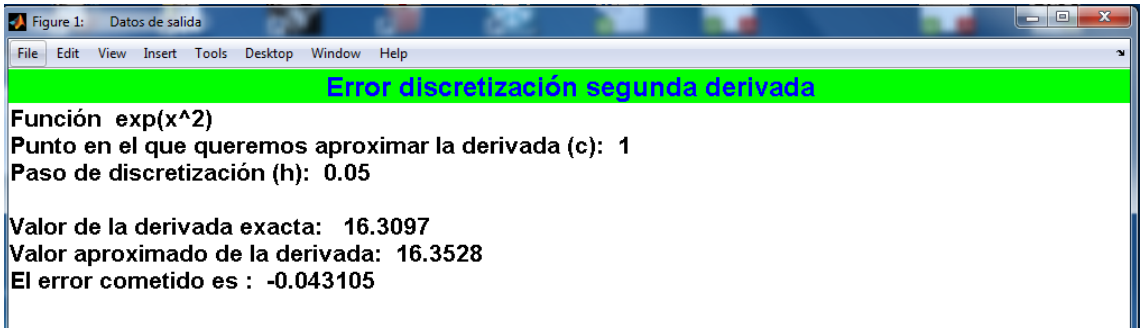


Figura 1.24

Vemos que el error reduce su orden un grado menos, por lo que realmente conseguimos una aproximación de orden $O(h^2)$.

9. Propagación directa de errores.

Seleccionamos el método en la ventana principal y nos aparecerán las variables que debemos introducir para llevar a cabo el método.

La función debe ser introducida como una función dependiente de variables con nomenclatura x1,x2,…, x10 como máximo.

En este caso queremos obtener el área S de un rectángulo para el que hemos obtenido las siguientes medidas de los lados;

$a = 2{,}0 \pm 0{,}1\ cm \qquad b = 5{,}0 \pm 0{,}2\ cm$

Introducimos los datos en nuestra interfaz;

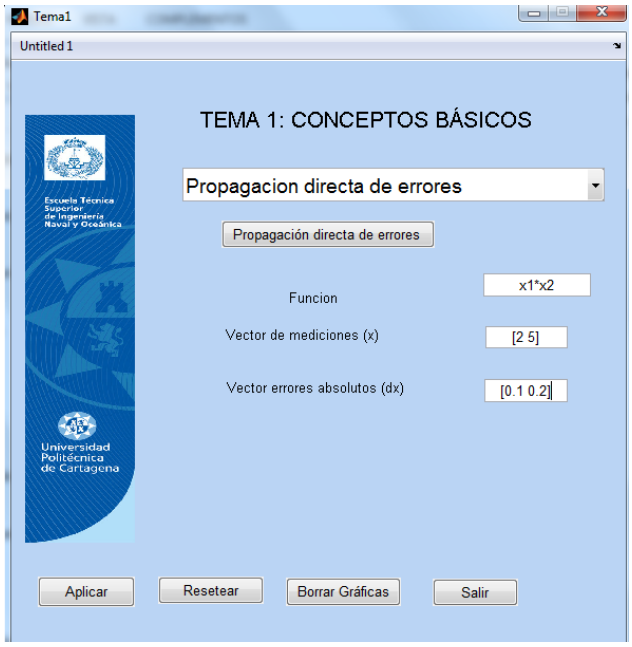


Figura 1.25

Al pulsar el botón aplicar, aparecerá una ventana emergente con los resultados;

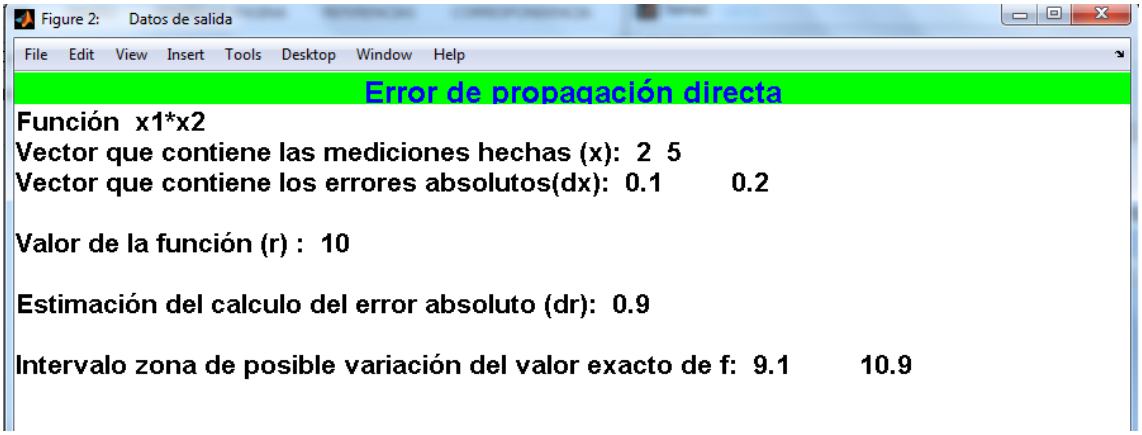


Figura 1.26

Por lo que el error absoluto del área obtenida será $dr = 0{,}9\ cm^2$, si utilizáramos unos instrumentos más precisos a la hora de tomar los datos, reduciríamos este error final, y al contrario.

10. Propagación inversa de errores.

Para comenzar, seleccionamos el método en la ventana principal y apareciendo las variables de entrada a introducir.

En este caso se desea medir la longitud de una circunferencia con un error no superior a 1cm. La medida del radio es 100cm, ¿qué instrumento hemos de emplear para medir el radio?

Introducimos los datos en la interfaz gráfica;

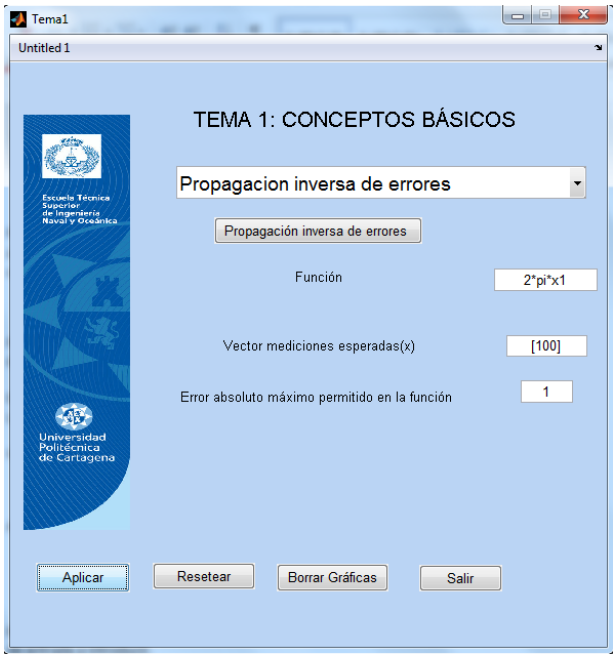


Figura 1.27

Al hacer doble click en el botón aplicar aparecen los resultados en una ventana emergente.

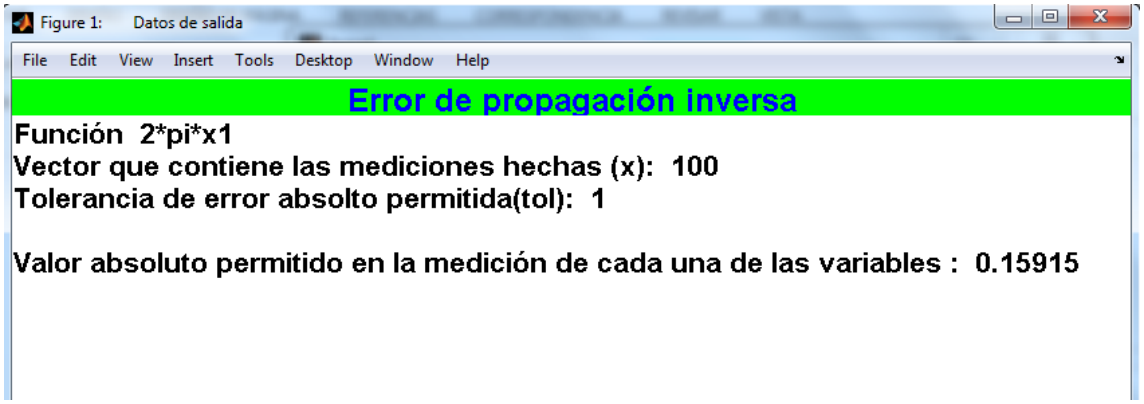


Figura 1.28

Por lo que tomamos $dr = 0.1\ cm$ que será el caso más desfavorable, pues el elegir $dr = 0.2\ cm$ implicaría un dL mayor del exigido. El error máximo que podemos cometer al medir el radio es de $1\ mm$, por lo que podremos emplear una cinta o regla milimetrada.

11. Método de Horner.

Para comenzar, seleccionamos el método en la ventana principal y apareciendo las variables de entrada a introducir.

Si llamamos a p al vector que contiene los coeficientes del polinomio $P(x)$, c al valor donde queremos evaluar el polinomio y m al número de derivadas que queremos realizar y d al polinomio con los resultados de cada derivada que queremos realizar.

Probamos para m=3 derivadas, en el punto c=3;

$$P(x) = x^5 - 6x^4 + 8x^3 + 8x^2 + 4x - 4$$

Figura 1.29

Al hacer doble click en el botón aplicar, aparece una ventana emergente con los resultados.

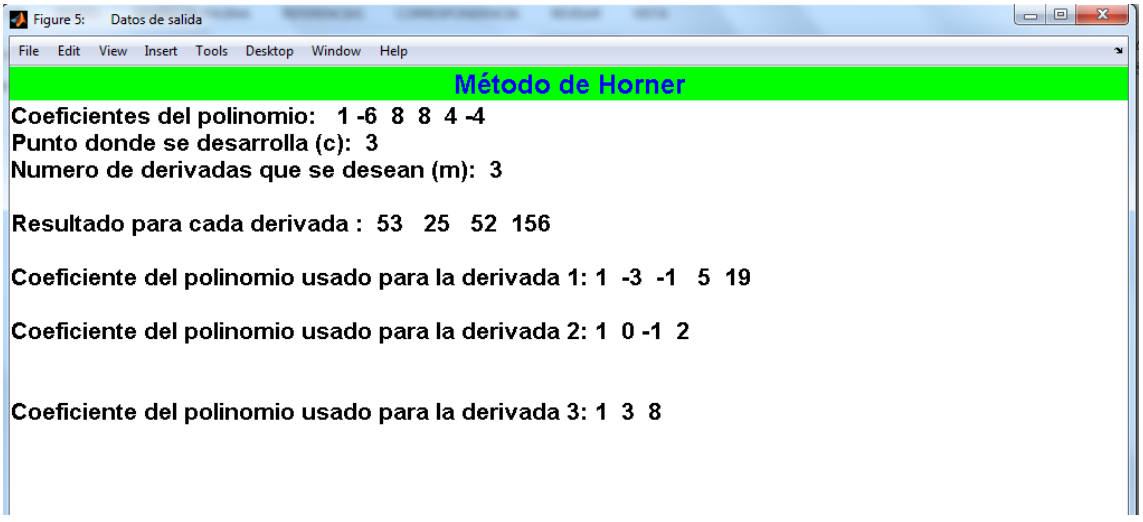


Figura 1.30

Es decir, el valor del polinomio en el punto c=3 será $P(3) = 53$.

El valor de la primera derivada en el punto c=3 será 25 y 52 y 156 serán los valores que toman la segunda y la tercera derivada respectivamente en el punto c=3.

Es fácil comprobar, por ejemplo, que el valor de la primera derivada obtenido es el correcto;

$$P'(x) = 5x^4 - 24x^3 + 24x^2 + 16x + 4$$

$$P'(3) = 5(3^4) - 24(3^3) + 24(3^2) + 16(3) + 4 = 25$$

12. Ecuación de segundo grado.

Consideremos por ejemplo

$$x^2 + (-10^5 - 10^{-5})x + 1 = 0$$

Que equivale a;

$$(x - 10^5)(x - 10^{-5}) = 0$$

Cuyas raíces exactas sabemos que son $x1 = 10^5 y\, x2 = 10^{-5}$

Figura 1.31

Pues bien, al hacer doble click en el botón aplicar podemos ver la diferencia entre realizar el cálculo por la forma usual o por la forma recomendada, y el error relativo cometido en ambos casos.

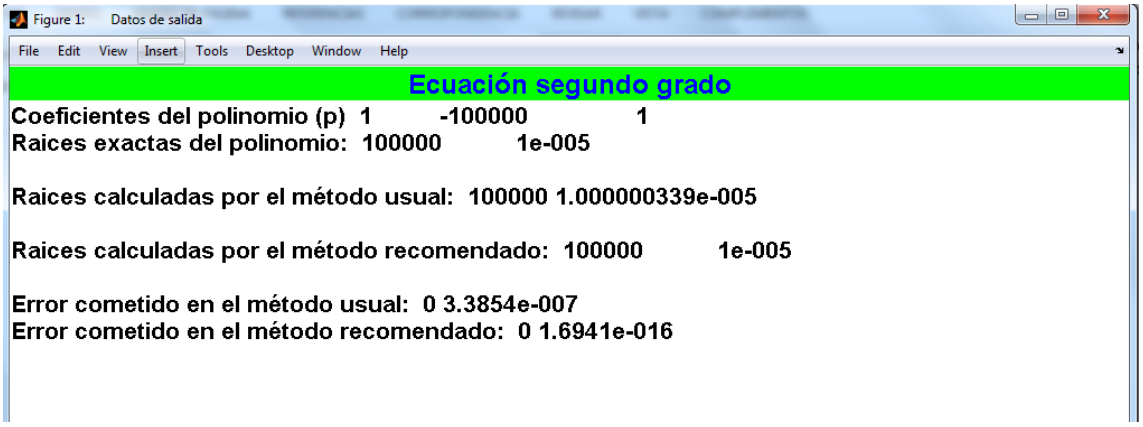


Figura 1.32

Conforme aumentamos el orden del coeficiente b podemos ver que la diferencia se hace cada vez más notable.

Por ejemplo para el polinomio;

$$(x - 10^6)(x - 10^{-6}) = 0$$

Cuyas raíces exactas sabemos que son $x1 = 10^6$ y $x2 = 10^{-6}$, obtenemos;

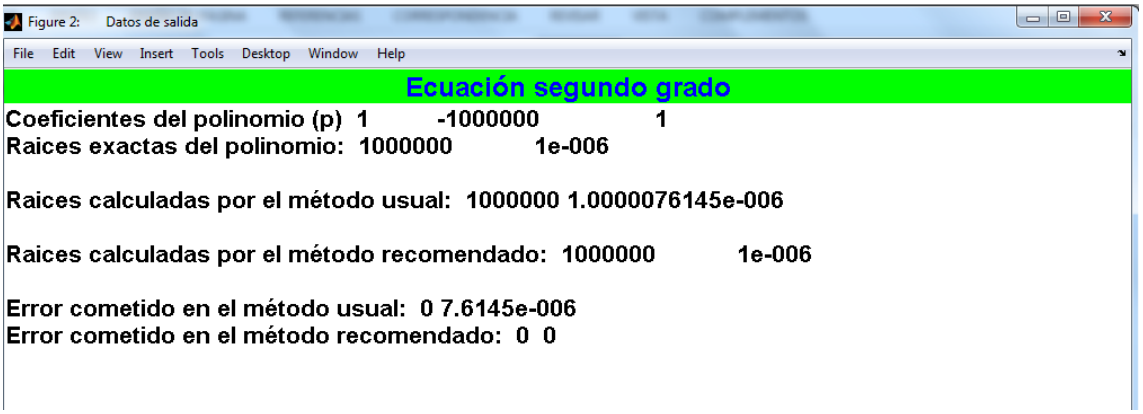


Figura 1.33

Como podemos observar, el error cometido en cálculo por el método recomendado se hace nulo.

13. Ejemplo de Wilkinson.

Seleccionamos el método en la pantalla principal de la interfaz, y nos aparecerán las variables de entrada a introducir.

En este caso, el algoritmo calcula las raíces en orden de mayor a menor, por lo que por ejemplo, si introducimos coe=2, nos referimos al coeficiente de x^{19}.

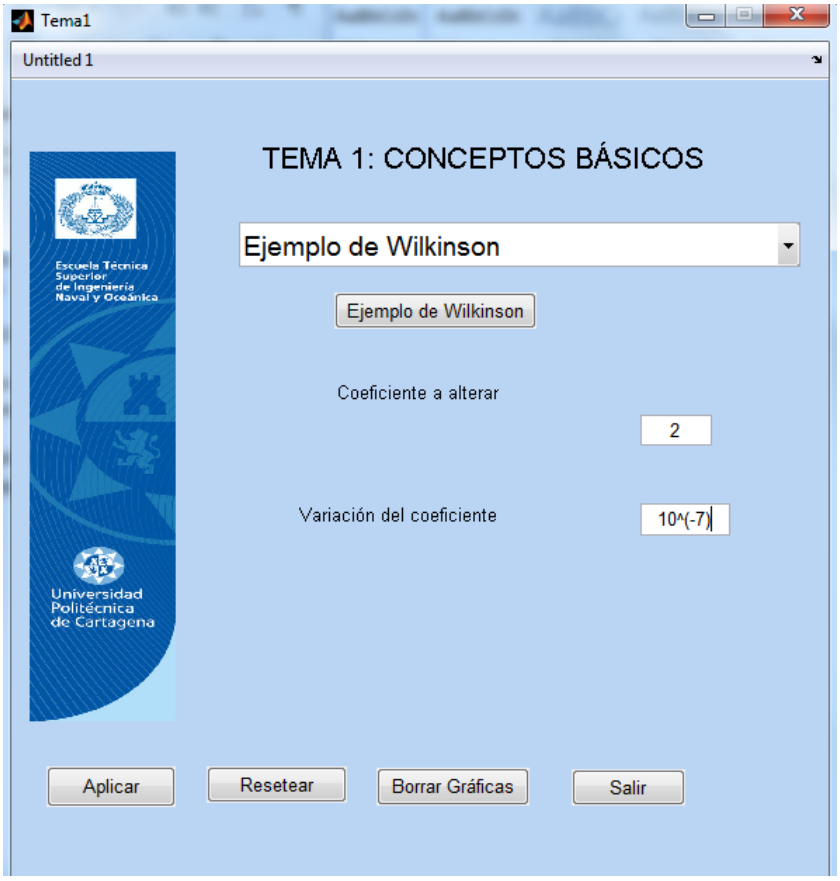


Figura 1.34

Al hacer doble click en el botón aplicar, aparece en una ventana emergente los resultados aproximados de las raíces con sus respectivos errores cometidos.

Ejemplo de Wilkinson

Coeficiente que va a ser alterado 2
Variación que se le añade al coeficiente (epsilon): 1e-007

Raices del polinomio alterado y errores cometidos:

20.422+0.999209i	-0.42198-0.99921i
20.422-0.999209i	-1.422+0.99921i
18.1573+2.47023i	-0.15728-2.4702i
18.1573-2.47023i	-1.1573+2.4702i
15.315+2.69876i	0.68504-2.6988i
15.315-2.69876i	-0.31496+2.6988i
12.8466+2.06273i	1.1534-2.0627i
12.8466-2.06273i	0.15342+2.0627i
10.9213+1.10372i	1.0787-1.1037i
10.9213-1.10372i	0.078725+1.1037i
9.5678	0.43217
9.1137	-0.11369
7.9941	0.005914
7.0002	-0.00023789
6	1.463e-006
5	4.1591e-007
4	-2.3407e-008
3	1.6846e-010
2	2.3435e-011
1	-3.8458e-013

Figura 1.35

Capítulo 2. Resolución numérica de ecuaciones y sistemas no lineales.

Introducción.

El objetivo de este capítulo es el desarrollo de una variedad de métodos que nos permitan calcular aproximaciones numéricas a las raíces de una ecuación o sistema de ecuaciones.

El proceso a seguir para determinar las raíces reales de una ecuación dada $f(x) = 0$ (algebraica o no) suele ser el siguiente, se comienza determinando un intervalo donde estén todas, es lo que se llama acotación de raíces. Seguidamente, se subdivide este intervalo en otros tales que cada uno contenga solamente una raíz, es lo que se denomina separación de raíces. Finalmente, se reduce la amplitud de dichos intervalos hasta que sus extremos constituyan valores aproximados aceptables, por exceso o por defecto, de la raíz contenida en su interior, en esto consiste la aproximación de raíces.

Comenzaremos recordando ciertos resultados de cálculo infinitesimal por su interés en la separación de raíces, sobre el que debemos resaltar el **teorema de Bolzano**, cuyas proposiciones son las siguientes:

- Sea $f: [a, b] \to R$ una función real, continua y tal que $f(a)f(b) < 0$ (es decir que toma valores opuestos en los extremos del intervalo), entonces existe $r \in (a, b) tal\ que\ f(r) = 0$.
- Si además se supone que f es derivable en el intervalo abierto (a,b) con $f'(x) > 0\ \forall x \in (a, b)$ $(o\ f'(x) < 0\ \forall x \in (a, b))$, entonces la raíz $r \in (a, b)$ de la ecuación $f(x) = 0$ es única.

- Un resultado sencillo que nos permite en muchos casos acotar el error absoluto que se comete al tomar el valor aproximado α de una raíz r viene dado por;

  Sea r una raíz de la ecuación $f(x) = 0$,α un valor aproximado de la misma, tal que $r, \alpha \in [a, b]$. Si f es derivable en [a,b] y $|f'(x)| \geq m1 > 0\ \forall x \in [a, b]$, entonces por el teorema del valor medio se tiene que $f(\alpha) - f(r) = (\alpha - r)f'(c)$ con c intermedio entre $\alpha\ y\ r$, luego $|f(\alpha) - f(r)| = |f(\alpha)| = |\alpha - r||f'(c)| \geq |\alpha - r|\, m1$, de donde se sigue que :

  $$|\alpha - r| \leq \frac{|f(\alpha)|}{m1} \quad (1)$$

  Donde m1 puede tomarse, por ejemplo, el $min_{x\in[a,b]}|f'(x)|$ siempre que este número sea positivo.

Ecuaciones no lineales con una variable.

Método de bisección de Bolzano.

En esta sección desarrollaremos nuestro primer método de localización para hallar ceros de funciones continuas. Debemos empezar con un intervalo de partida [a,b] en el que f(a) y f(b) tengan distinto signo. Entonces, por el teorema del valor medio, la gráfica $y = f(x)$ cruzará el eje OX en un cero $x = r$ que está en dicho intervalo. El método de bisección consiste en ir acercando sistemáticamente los extremos del intervalo hasta que obtengamos un intervalo de anchura suficientemente pequeña en el que se localiza un cero. El proceso de decisión para subdividir el intervalo cosiste en tomar el punto medio del intervalo $c = (a + b)/2$ y luego analizar las tres posibilidades que pueden darse:

1. Si f(a) y f(c) tienen signos opuestos, entonces hay un cero en [a,c].
2. Si f(c) y f(b) tienen signos opuestos, entonces hay un cero en [c,b].
3. Si f(c)=0, entonces c es un cero.

Si ocurre bien el caso (1) o bien el caso (2) entonces hemos encontrado un intervalo la mitad de ancho que el original que contiene una raíz. Para continuar el proceso, renombramos el nuevo intervalo más pequeño también como [a,b] y repetimos el proceso hasta que el intervalo sea tan pequeño como deseemos. Utilizaremos la siguiente nomenclatura:

$[a_0, b_0]$ es el intervalo de partida y $c_0 = \frac{a_0+b_0}{2}$ es su punto medio

$[a_1, b_1]$ es el intervalo en el que se localiza un cero y c_1 es su punto medio; el intervalo $[a_1, b_1]$ es la mitad de ancho que $[a_0, b_0]$.

Después de llegar al intervalo $[a_n, b_n]$, en el que también se localiza un cero y cuyo punto medio es c_n, se construye el intervalo $[a_{n+1}, b_{n+1}]$, en el que también se sigue localizando un cero, y que mide la mitad que $[a_n, b_n]$.

Convergencia del método de bisección. Supongamos que $f \in C[a, b]$ y que $f(a) y\, f(b)$ tienen signos distintos. Sea $\{c_n\}_{n=0}^{\infty}$ la sucesión de puntos medios de los intervalos generados por el método de bisección. Entonces existe un número $r \in [a, b]$ tal que $f(r) = 0$ y, además,

$$|r - c_n| \leq \frac{b-a}{2^{n+1}} para\ n = 0,1, \ldots, \quad (2)$$

En particular, la sucesión $\{c_n\}_{n=0}^{\infty}$ converge al cero $x = r$, esto es;

$$\lim_{n\to\infty} c_n = r.$$

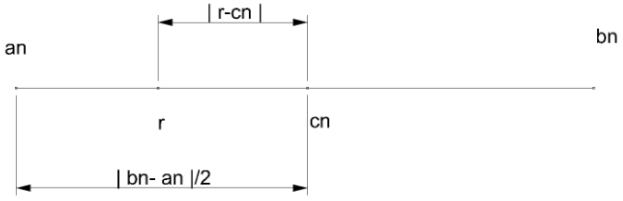


Figura 2.0

Demostración. Observemos que las sucesivas anchuras de los intervalos se ajustan al siguiente patrón:

$$b_1 - a_1 = \frac{b_0 - a_0}{2^1} \quad (3)$$

$$b_2 - a_2 = \frac{b_1 - a_1}{2} = \frac{b_0 - a_0}{2^n}. \quad (4)$$

Entonces, usando inducción matemática,

$$b_n - a_n = \frac{b_0 - a_0}{2^n}. \quad (5)$$

Ahora, la sucesión (a_n) es creciente y está acotada superiormente, luego tiene límite a; la sucesión (b_n) es decreciente y está acotada inferiormente, luego tiene límite b. Tomando límite cuando $n \to \infty$ en (5) deducimos que $a\ y\ b$ deben coincidir sea $r = b = a$. Por continuidad tenemos que $f(r) = 0$, luego r es un cero de f que está en cada intervalo $[a_n, b_n]$. Como el punto medio c_n también está en el intervalo $[a_n, b_n]$, la distancia entre c_n y r no puede ser mayor que la mitad de la anchura de dicho intervalo. En consecuencia,

$$|r - c_n| \leq \frac{b_n - a_n}{2} \quad para\ todo\ n \qquad (6)$$

Combinando (5) y (6), resulta

$$|r - c_n| \leq \frac{b_0 - a_0}{2^n} \quad para\ todo\ n \qquad (6)$$

Lo que implica que la sucesión $\{c_n\}_{n=0}^{\infty}$ converge a r, lo que completa la demostración.

Teorema del punto fijo: método iterativo general.

En esta sección estudiaremos el proceso de iteración que consiste en sustituir repetidamente en una misma fórmula el valor previamente obtenido.

Necesitamos una regla, formula o función $g(x)$, con la que calcularemos los sucesivos términos, junto con un valor de partida p_0. Lo que se produce es una sucesión de valores $\{p_k\}$ obtenida mediante el proceso iterativo $p_{k+1} = g(p_k)$. La sucesión se ajusta al siguiente patrón.

$$p_0$$

$$p_1 = g(p_0)$$

$$p_2 = g(p_1)$$

....

$$p_k = g(p_{k-1})$$

$$p_{k+1} = g(p_k)$$

...

Un punto fijo de una función $g(x)$ es un número real P tal que $P = g(P)$.

Geométricamente hablando, los puntos fijos de una función $g(x)$ son los puntos de intersección de la curva $y = g(x)$ con la recta $y = x$.

La iteración $p_{n+1} = g(p_n)$ para $n = 0,1, \ldots$, se llama iteración de punto fijo.

Supongamos que g es una función continua y que $\{p_n\}_{n=0}^{\infty}$ es una sucesión generada por iteración de punto fijo. Si $lim_{n\to\infty} p_{n+1} = P$, entonces P es un punto fijo de $g(x)$.

Demostración. Si $\lim_{n\to\infty} p_n = P$, entonces $\lim_{n\to\infty} p_{n+1} = P$. Usando esto, la continuidad de g y la relación $p_{n+1} = g(p_n)$, deducimos que

$$g(P) = g\left(\lim_{n\to\infty} p_n\right) = \lim_{n\to\infty} g(p_n) = \lim_{n\to\infty} p_{n+1} = P. \qquad (7)$$

Por tanto, P es un punto fijo de $g(x)$.

Los dos siguientes teoremas establecen condiciones para la existencia de un punto fijo y para la convergencia del proceso de iteración del punto fijo.

Supongamos que $g \in C[a, b]$.

(1) Si la imagen de la aplicación $y = g(x)$ verifica que $y \in [a, b]$ para cada punto $x \in [a, b]$, entonces g tiene un punto fijo en $[a, b]$.
(2) Supongamos, además, que $g'(x)$ está definida en (a, b) y que $|g'(x)| < 1$ para todo $x \in (a, b)$, entonces g tiene un único punto fijo en P en $[a, b]$.

Demostración. Ahora debemos probar que la solución es única. Supongamos, por el contrario, que existen dos puntos fijos distintos $P_1 y\, P_2$. Aplicando el teorema del valor medio, deducimos que existe un número $d \in (a, b)$ tal que

$$g'(d) = \frac{g(P_2) - g(P_1)}{P_2 - P_1} \qquad (8)$$

A continuación, usamos $g(P_1) = P_1$ y que $g(P_2) = P_2$ para obtener,

$$g'(d) = \frac{P_2 - P_1}{P_2 - P_1} = 1, \qquad (9)$$

Lo que contradice la hipótesis hecha en (2) de que $|g'(x)| < 1$ en (a, b). Así que no es posible que existan dos puntos fijos y, por tanto, bajo la condición dada en (2), $g(x)$ tiene un único punto fijo P en $[a, b]$.

Vamos ahora a establecer el teorema que se suele usar para determinar si el proceso de iteración de punto fijo produce una sucesión convergente o divergente.

Supongamos que $g, g' \in C[a,b], K$ es una constante positiva, $p_0 \in (a,b)$ y $g(x) \in [a,b]$ para todo $x \in [a,b]$. Entonces hay un punto fijo P de g en $[a,b]$.

(3) Si $|g'(x)| \leq K < 1$ para todo $x \in [a,b]$, entonces P es el único punto fijo de g en $[a,b]$ y la iteración $p_n = g(p_{n-1})$ converge a un punto fijo P. En este caso, se dice que P es un punto fijo atractivo.

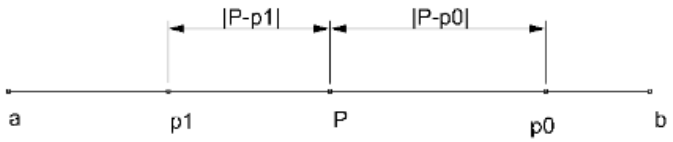


Figura 2.1

Puede tomarse como $K = max_{x\in[a,b]}|g'(x)|$ siempre que este K sea menor que 1.

(4) Si $|g'(P)| > 1\ y\ p_0 \neq P$ entonces la iteración $p_n = g(p_{n-1})$ no converge a P. En este caso, se dice que P es un punto fijo repulsivo y la iteración presenta divergencia local.

(5) Puede probarse sin dificultad que los errores en el paso enésimo verifican las acotaciones:

$$|P - p_n| \leq \frac{k}{1-k}|p_n - p_{n-1}| \leq \frac{k^n}{1-k}|p_1 - p_0| \qquad (10)$$

La primera nos sirve para estimar el error a posteriori y la última para estimarlo a priori y determinar el número de iteraciones a realizar para garantizar una aproximación querida.

Método de LaGrange o regula falsi.

Una de las razones de su introducción es que la velocidad de convergencia del método de bisección es bastante baja. Como antes, supongamos que $f(a)$ y $f(b)$ tienen distinto signo. En el método de bisección se usa el punto medio del intervalo [a,b] para llevar a cabo el siguiente paso. Suele conseguirse una aproximación mejor usando el punto $(c,0)$ en el que la recta secante L que pasa por los puntos $(a,f(a))$ y $(b,f(b))$ cruza el eje OX. Para hallar el punto c, igualamos dos fórmulas para la pendiente m de la recta L:

$$m = \frac{f(b) - f(a)}{b - a} \qquad (11)$$

Que resulta de usar los puntos $(a, f(a))$ y $(b, f(b))$, y

$$m = \frac{0 - f(b)}{c - b} \qquad (12)$$

Que resulta de usar los puntos $(c, 0)$ y $(b, f(b))$.

Igualando las pendientes que aparecen en (11) y (12), tenemos,

$$\frac{f(b) - f(a)}{b - a} = \frac{0 - f(b)}{c - b}$$

De donde c puede despejarse fácilmente, obteniéndose

$$c = b - \frac{f(b)(b - a)}{f(b) - f(a)}. \qquad (13)$$

Las tres posibilidades son las mismas de antes:

- Si $f(a) y\ f(c)$ tienen distinto signo, entonces hay un cero en $[a, c]$.
- Si $f(c) y\ f(b)$ tienen distinto signo, entonces hay un cero en $[c, b]$.
- Si $f(c) = 0$, entonces c es un cero de f.

La fórmula (13) se utiliza para construir una sucesión de intervalos $\{[a_n, b_n]\}$ cada uno de los cuales contiene un cero. En cada paso la aproximación al cero obtenida es:

$$c_n = b_n - \frac{f(b_n)(b_n - a_n)}{f(b_n) - f(a_n)}, \qquad (14)$$

Y puede probarse que la sucesión $\{c_n\}$ converge a un cero r de la función. Sin embargo, hay que prestar atención al siguiente hecho: aunque la anchura del intervalo $b_n - a_n$ se hace más pequeña, es posible que no tienda a cero; si la curva $y = f(x)$ es convexa cerca de $(r, 0)$, entonces uno de los extremos $a_n\ o\ b_n$ permanece estacionario y el otro tiende a la solución.

El criterio de parada usado en el método de bisección no es útil para el método de la *regula falsi* porque podría producir iteraciones sin fin. En este caso se usan tanto la cercanía entre sí de dos aproximaciones sucesivas $|c_n - c_{n-1}|$ como el tamaño $|f(c_n)|$.

Los métodos de Newton-Raphson y de la secante.

Métodos tangenciales para el cálculo de raíces.

Si $f(x)$, $f'(x)$ y $f''(x)$ son continuas cerca de una raíz p, esta información adicional sobre la naturaleza de $f(x)$ puede utilizarse para desarrollar algoritmos que produzcan sucesiones $\{p_k\}$ que converjan a p más rápidamente que el método de bisección o en el de la *regula falsi*. El

método de Newto-Raphson, que descansa en la continuidad de $f'(x)$ y $f''(x)$, es uno de los algoritmos más útiles y mejor conocidos.

Supongamos que la aproximación inicial p_0 está cerca de la raíz p. Entonces la curva $y = f(x)$ y el eje de abscisas se cortan en el punto $(p, 0)$ y, además, el punto $(p_0, f(p_0))$ está situado en la curva cerca de $(p, 0)$. Definimos p_1 como el punto de intersección del eje de abscisas con la recta tangente de a la curva en el punto $(p_0, f(p_0))$, p_1 estará, en este caso, más cerca de p que p_0. Podemos encontrar la ecuación que relaciona p_1 con p_0 igualando dos fórmulas distintas para la pendiente m de la recta tangente. Por un lado,

$$m = \frac{0 - f(p_0)}{p_1 - p_0}, \qquad (15)$$

Que es la pendiente de la línea recta que pasa por $(p_1, 0)$ y $(p_0, f(p_0))$; por otro lado,

$$m = f'(p_0) \qquad (16)$$

Que es la pendiente de la recta tangente a la curva en el punto $\big(p_0, f(p_0)\big)$. Igualando los valores de la pendiente m en las fórmulas $(15) y\ (16)$ y despejando p_1 obtenemos,

$$p_1 = p_0 - \frac{f(p_0)}{f'(p_0)} \qquad (17)$$

Este proceso puede repetirse para obtener una sucesión $\{p_k\}$ que converge a p. Hagamos más precisas estas ideas.

Teorema de Newton-Raphson.

Supongamos que la función $f \in C^2[a, b]$ y que existe un número $p \in [a, b]$ tal que $f(p) = 0$. Si $f'(p) \neq 0$, entonces existe $\delta > 0$ tal que la sucesión $\{p_k\}_{k=0}^{\infty}$ definida por el proceso iterativo

$$p_k = g(p_{k-1}) = p_{k-1} - \frac{f(p_{k-1})}{f'(p_{k-1})} \quad para\ k = 1,2,.. \qquad (18)$$

Converge a p cualquiera que sea la aproximación inicial $p_0 \in [p - \delta, p + \delta]$.

La función $g(x)$ definida por la relación

$$g(x) = x - \frac{f(x)}{f'(x)} \qquad (19)$$

Se llama función de iteración de Newton-Raphson. Puesto que $f(p) = 0$, es fácil ver que $g(p) = p$, lo que nos dice que la iteración de Newton-Raphson para hallar una raíz de la ecuación $f(x) = 0$ consiste en hallar un punto de $g(x)$.

El método de la secante.

En el algoritmo de Newton-Raphson hay que evaluar dos funciones en cada iteración, $f(p_{k-1})$ y $f'(p_{k-1})$. Tradicionalmente, el cálculo de la derivada de una función elemental puede llegar a suponer un esfuerzo considerable. Sim embargo, con los modernos paquetes de cálculo simbólico, esto no es un problema serio hoy en día. Aun así, hay muchas funciones dadas de forma no elemental (como integrales o sumas de series, etc.) para las que sería deseable disponer de un método que converja casi tan rápidamente como el de Newton-Raphson y que necesite únicamente evaluaciones de $f(x)$ y no de $f'(x)$. El método de la secante necesita sólo una evaluación en $f(x)$ por paso y una raíz simple tiene un orden de convergencia $R \approx 1.618033989$; es casi tan veloz como el de Newton, cuyo orden de convergencia es 2.

La fórmula de iteración del método de la secante es la misma que la que aparece en el método de *regula falsi,* la diferencia entre ambos estriba en la estructura lógica de la forma de decidir cómo se elige el siguiente término. Partimos de dos puntos iniciales $(p_0, f(p_0))$ y $(p_1, f(p_1))$ cercanos al punto $(p, 0)$, y se define p_2 como la abscisa del punto de intersección de la recta que pasa por estos dos puntos con el eje OX.

La fórmula que relaciona $p_2, p_1\ y\ p_0$ se halla escribiendo la pendiente de la recta en cuestión:

$$m = \frac{f(p_1) - f(p_0)}{p_1 - p_0} \quad y \quad m = \frac{0 - f(p_1)}{p_2 - p_1} \qquad (20)$$

El primer valor de m en (20) es la pendiente de la recta secante que pasa por dos puntos iniciales y el segundo valor es la pendiente de la recta que pasa por $(p_1, f(p_1))$y $(p_2, 0)$. Igualando los miembros derechos de las dos fórmulas y despejando $p_2 = g(p_1, p_0)$ obtenemos,

$$p_2 = g(p_1, p_0) = p_1 - \frac{f(p_1)(p_1 - p_0)}{f(p_1) - f(p_0)}. \qquad (21)$$

El término general de la sucesión generada por este método viene dado por la fórmula de iteración de dos puntos

$$p_{k+1} = g(p_k, p_{k-1}) = p_k - \frac{f(p_k)(p_k - p_{k-1})}{f(p_k) - f(p_{k-1})}. \qquad (21)$$

El método necesita pues de dos valores $p_0\ y\ p_1$ para iniciarse, que pueden ser los extremos del intervalo que contiene la raíz buscada, o bien partiendo de un p_0 inicial y calcular p_1 por algún otro método (por ejemplo aplicando el método de Newton).

Orden de convergencia de un método.

Además de la convergencia de un método dado, se quiere saber si la sucesión definida por las iteraciones $p_{k+1} = g(p_k)$ converge rápidamente a la raíz buscada p; es decir como disminuye el error $e_k = p_k - p$ de una iteración a la siguiente.

Para estudiar el orden de un método, definido por la función de iteración g, si esta es suficientemente regular, se recurre al desarrollo de Taylor, de modo que si $g \in C^{(k+1)}(II)$ en un intervalo (II) que contiene a p se tendrá:

$$e_{k+1} = p_{k+1} - p = g(p_k) - g(p) =$$

$$g'(p)e_k + \frac{1}{2}g''(p)e_k^2 + \cdots + \frac{g^{(n)}(p)}{n!}e_k^n + \frac{g^{(n+1)}(\varepsilon_k)}{(n+1)!}e_k^{n+1} \qquad (22)$$

Por tanto $\lim_{k\to\infty} \frac{|e_{k+1}|}{|e_k|} = |g'(p)|$, luego si $0 < |g'(p)| < 1$ se dice que el método tiene convergencia lineal o de primer orden. Si $g'(p) = 0$. Se deduce que $\lim_{k\to\infty} \frac{|e_{k+1}|}{|e_k|^2} = \frac{1}{2}|g''(p)|$, luego si $g''(p) \neq 0$ se dice que la convergencia es cuadrática o de segundo orden. Y, en general, si se verifican las condiciones $g'(p) = g''(p) = \cdots = g^{(r-1)}(p) = 0$ y $g^r(p) \neq 0$ el método será de orden r.

En general, si se verifican las igualdades $g'(p) = g''(p) = \cdots = g^{(r-1)}(p) = 0$, pero no sabemos si $g^r(p)$ se anula o no, decimos que la convergencia es de orden, al menos, r.

Convergencia global y local.

Los métodos de localización dependen de la determinación de un intervalo inicial $[a, b]$ en el que $f(a)$ y $f(b)$ tengan distinto signo. Un vez encontrado este intervalo, no importa lo grande que sea, podremos empezar a iterar hasta que encontremos una raíz con la precisión deseada. Por esta razón se dice que estos métodos son globalmente convergentes. Sin embargo, si $f(x) = 0$ tiene varias raíces en $[a, b]$, entonces debemos encontrar un intervalo de partida distinto para hallar cada raíz y no suele ser fácil hallar estos intervalos más pequeños en los que el signo de $f(x)$ cambia.

Los métodos de Newton-Raphson y de la secante para resolver $f(x) = 0$ requieren, como garantía de su convergencia, que el punto inicial esté cerca de la raíz, por lo que se dice que son localmente convergentes. A cambio, suelen converger más rápidamente que los métodos globales, de manera que existen algoritmos híbridos que empiezan con un método de convergencia global y, cuando las iteraciones nos han llevado cerca de la raíz, cambian a un método de convergencia local.

1. Teorema de la convergencia local.

Sea f'' continua y f' no nula en algún intervalo abierto que contenga la raíz p de $f(x) = 0$. Entonces, existe $\varepsilon > 0$ tal que el método de Newton es convergente para todo p_0 tal que $|p_0 - p| \leq \varepsilon$. Si además f''' es continua la convergencia es, al menos, cuadrática.

2. Teorema de la convergencia global-1.

Sea $f \in C^2([a, b])$ verificando:

1. $f(a) \cdot f(b) < 0$.
2. $\forall\, x \in [a, b] es\ f'(x) \neq 0$ $(monotomía\ estricta)$.
3. $\forall\, x \in [a, b] es\ f''(x) \geq 0$ $(o\ \forall\, x \in [a, b] es\ f''(x) \leq 0$, concavidad en el mismo sentido)

Entonces, existe una única raíz p de $f(x) = 0$ en $[a, b]$ y la sucesión $\{p_n\}_0^\infty$, definida por el algoritmo del método de Newton converge, hacia p para todo $x_0 \in [a, b]$ tal que $f(x_0) \cdot f''(x_0) \geq 0$. Si además $f \in C^{(3)}([a, b])$ la convergencia es, al menos, cuadrática.

3. Teorema de la convergencia global-2.

Sea $f \in C^2([a, b])$ verificando:

1. $f(a) \cdot f(b) < 0$.
2. $\forall\, x \in [a, b] es\ f'(x) \neq 0$ $(monotomía\ estricta)$.
3. $\forall\, x \in [a, b] es\ f''(x) \geq 0$ $(o\ \forall\, x \in [a, b] es\ f''(x) \leq 0$, concavidad en el mismo sentido).
4. $\max\left|\frac{f(a)}{f'(a)}\right|, \left|\frac{f(b)}{f'(b)}\right| \leq b - a$.

Entonces, existe una única raíz p de $f(x) = 0$ en $[a, b]$ y la sucesión $\{p_n\}_0^\infty$, definida por el algoritmo del método de Newton converge, hacia p para todo $x_0 \in [a, b]$. Si además $f \in C^{(3)}([a, b])$ la convergencia es, al menos, cuadrática.

Aceleración de la convergencia. Métodos de Aitken y Steffensen.

Una técnica conocida como método Δ^2 de Aitken permite acelerar la convergencia de cualquier sucesión que sea linealmente convergente. Para presentar esta técnica necesitamos una definición previa.

Dada una sucesión $\{p_n\}_{n=0}^\infty$, se define la diferencia progresiva Δp_n mediante

$$\Delta p_n = p_{n+1} - p_n \ \ para\ n \geq 0 \qquad (23)$$

Las potencias superiores $\Delta^k p_n$ se definen de manera recursiva mediante

$$\Delta^k p_n = \Delta^{k-1}(\Delta p_n) \ \ para\ k \geq 2 \qquad (24).$$

Aceleración de Aitken.

Sea $\{p_n\}_{n=0}^\infty$ una sucesión que converge linealmente a su límite p y tal que $p - p_n \neq 0$ para todo $n \geq 0$. Si existe un número real A con $|A| < 1$ y tal que

$$\lim_{n\to\infty} \frac{p - p_{n+1}}{p - p_n} = A, \qquad (25)$$

Entonces la sucesión $\{q_n\}_{n=0}^\infty$ definida por

$$q_n = p_n - \frac{(\Delta p_n)^2}{\Delta^2 p_n} = p_n - \frac{(p_{n+1} - p_n)^2}{p_{n+2} - 2p_{n+1} + p_n} \qquad (26)$$

Converge a p más rápidamente que $\{p_n\}_{n=0}^{\infty}$ en el sentido de que

$$\lim_{n\to\infty} \left| \frac{p - q_{n+1}}{p - q_n} \right| = 0, \qquad (27)$$

Cuando el método de Aitken se combina con una iteración de punto fio, el resultado se conoce como método de aceleración de Steffensen.

Acotación de raíces de una ecuación polinómica.

Los métodos anteriormente expuestos son aplicables a la resolución de ecuaciones polinómicas $p(x) = 0$, siendo

$$p(x) = a_0x^n + a_1x^{n-1} + \cdots + a_{n-1}x + a_n = 0 \; con \; a_i \in R \; y a_0 \neq 0$$

Pero hay diversas peculiaridades relativas a la evaluación de polinomios y sus derivadas, así como la acotación de sus raíces, que por su eficiencia pasamos a relatar.

En general, dada una ecuación $f(x) = 0$, se dice que el número real L es la cota superior de sus raíces reales si para toda raíz real r de dicha ecuación se verifica que $r \leq L$. Análogamente se dice que el número real l es una cota inferior de sus raíces si verifica que $l \leq r$. Y se dice que se han acotado sus raíces si se ha determinado un intervalo $[l, L]$ que contiene todas las raíces. Si se admite la posibilidad de raíces complejas, se dirá que se han acotado si existe $M \geq 0$ tal que $|r| \leq M$ para toda raíz r de $f(x) = 0$. Veamos algunos resultados sencillos de acotación de ecuaciones polinómicas.

Sea la ecuación polinómica $p(x) = a_0x^n + a_1x^{n-1} + \cdots + a_{n-1}x + a_n = 0 \; con \; a_0 \neq 0$, entonces si $\lambda = \max\{\left|\frac{a_i}{a_0}\right| \; 1 \leq i \leq n\}$ toda raíz de $p(x) = 0$ verifica que $|r| \leq \lambda + 1$. Es decir todas las raíces de la ecuación están contenidas en el círculo de radio $\lambda + 1$.

Veremos a continuación un método para obtener una cota superior L de las raíces reales de una ecuación polinómica. Puesto que las ecuaciones $f(x) = 0$ y $f(-t) = 0$ tienen raíces opuestas, si L' es una cota superior de las raíces de $f(-t) = 0$ entonces $l = -L'$ es una cota inferior de las raíces reales de $f(x) = 0$, en consecuencia si sabemos hallar una cota superior también sabemos obtener una cota inferior.

Método de Laguerre-Thibault.

Si $L \geq 0$ y en la división de $p(x)$ por $x - L$ son no negativos todos los coeficientes del cociente y también el resto, entonces L es una cota superior de las raíces reales de $p(x)$.

Demostración. En efecto, $p(x) = p(L) + (x - L)(b_0x^{n-1} + \cdots + b_{n-2}x + b_{n-1})$ y si $p(L) \geq 0$ y $b_i \geq 0$ $(i = 0,1, \ldots, n-1)$, entonces para todo $x > L \geq 0$ real resulta ser que $p(x) > 0$ y x no puede ser raíz, de donde se sigue el resultado.

El método de Newton-Raphson para sistemas no lineales.

Vamos a construir el método de Newton-Raphson en el caso bidimensional; construcción que se generaliza fácilmente a dimensiones mayores.

Consideremos el sistema

$$u = f_1(x, y)$$
$$v = f_2(x, y) \qquad (28)$$

Que puede verse como una transformación del plano XOY en el plano UOV. Si estamos interesados en el comportamiento de esta transformación cerca del punto (x_0, y_0), cuya imagen es el punto (u_0, v_0), y si las dos funciones tienen derivadas parciales continuas, entonces podemos usar la diferencial del sistema para escribir un sistema de aproximaciones incrementales lineales válidas cerca del punto (x_0, y_0) en cuestión:

$$u - u_0 \approx \frac{\partial}{\partial x} f_1(x_o, y_0)(x - x_0) + \frac{\partial}{\partial y} f_1(x_0, y_0)(y - y_0)$$

$$v - v_0 \approx \frac{\partial}{\partial x} f_2(x_o, y_0)(x - x_0) + \frac{\partial}{\partial y} f_2(x_0, y_0)(y - y_0) \qquad (29)$$

El sistema (29) es una aproximación lineal local que nos da una idea del efecto que pequeños cambios en las variables independientes producen en las variables dependientes. Si usamos la matriz jacobiana $\boldsymbol{J}(x_0, y_0)$, esta relación se escribe de forma más cómoda como

$$\begin{bmatrix} u - u_0 \\ v - v_0 \end{bmatrix} \approx \begin{bmatrix} \frac{\partial}{\partial x} f_1(x_0, y_0) & \frac{\partial}{\partial y} f_1(x_0, y_0) \\ \frac{\partial}{\partial x} f_2(x_0, y_0) & \frac{\partial}{\partial y} f_2(x_0, y_0) \end{bmatrix} \begin{bmatrix} x - x_0 \\ y - y_0 \end{bmatrix} \qquad (30)$$

Si escribimos el sistema (28) como una función vectorial $\boldsymbol{V} = \boldsymbol{F}(\boldsymbol{X})$, entonces la matriz jacobiana $\boldsymbol{J}(x, y)$ es al análogo bidimensional de la derivada, porque la relación (30) queda

$$\Delta \boldsymbol{F} \approx \boldsymbol{J}(x_0, y_0) \Delta \boldsymbol{X} \qquad (31)$$

Usaremos la aproximación (31) para demostrar el método de Newton bidimensional. Consideremos el sistema de ecuaciones que resulta de igualar $u\ y\ v$ a cero en (28):

$$0 = f_1(x, y)$$
$$0 = f_2(x, y) \qquad (32)$$

Supongamos que (p, q) es una solución de (32); es decir,

$$0 = f_1(p, q)$$
$$0 = f_2(p, q) \qquad (33)$$

Si consideramos pequeños cambios en las funciones cerca del punto inicial (p_0, q_0) próximo a la solución (p, q):

$$\begin{matrix} \Delta u = u - u_0 & \Delta p = x - p_0 \\ \Delta v = v - v_0 & \Delta q = y - q_0 \end{matrix} \qquad (34)$$

Ponemos $(x, y) = (p, q)$ en (28) y usamos (33), de manera que $(u, v) = (0,0)$, entonces los cambios en las variables dependientes son

$$\begin{matrix} u - u_0 = f_1(p, q) - f_1(p_0, q_0) = 0 - f_1(p_0, q_0) \\ v - v_0 = f_2(p, q) - f_2(p_0, q_0) = 0 - f_2(p_0, q_0) \end{matrix} \qquad (35)$$

Ahora usamos los resultados de (35)en la aproximación lineal (30) y obtenemos

$$\begin{bmatrix} \frac{\partial}{\partial x} f_1(p_0, q_0) & \frac{\partial}{\partial y} f_1(p_0, q_0) \\ \frac{\partial}{\partial x} f_2(p_0, q_0) & \frac{\partial}{\partial y} f_2(p_0, q_0) \end{bmatrix} \begin{bmatrix} \Delta p \\ \Delta q \end{bmatrix} \approx - \begin{bmatrix} f_1(p_0, q_0) \\ f_2(p_0, q_0) \end{bmatrix}. \qquad (36)$$

Si la matriz jacobiana $\boldsymbol{J}(p_0, q_0)$ que aparece en (36) es invertible, entonces podemos despejar $\Delta \boldsymbol{P} = [\Delta p \;\; \Delta q]' = [p - q]' - [p_0 \;\; q_0]'$ de manera que

$$\Delta \boldsymbol{P} \approx -\boldsymbol{J}(p_0, q_0)^{-1} \boldsymbol{F}(p_0, q_0). \qquad (37)$$

Esto nos proporciona la siguiente aproximación $\boldsymbol{P}_1$ a la solución $\boldsymbol{P} = [p \; q]$:

$$\boldsymbol{P}_1 = \boldsymbol{P}_0 + \Delta \boldsymbol{P} = \boldsymbol{P}_0 - \boldsymbol{J}(p_0, q_0)^{-1} \boldsymbol{F}(p_0, q_0). \qquad (38)$$

Hagamos notar que la fórmula (38) es la generalización de la fórmula de iteración del método de Newton-Raphson para funciones de una variable que, como vimos, es

$$p_1 = p_0 - \frac{f(p_0)}{f'(p_0)}.$$

Esquema del método de Newton-Raphson.

Supongamos que hemos obtenido $\boldsymbol{P}_K$.

Paso 1. Evaluamos la función

$$\boldsymbol{F}(\boldsymbol{P}_k) = \begin{bmatrix} f_1(p_k, q_k) \\ f_2(p_k, q_k) \end{bmatrix}.$$

Paso 2. Evaluamos la matriz jacobiana

$$\boldsymbol{J}(\boldsymbol{P}_k) = \begin{bmatrix} \dfrac{\partial}{\partial x} f_1(p_k, q_k) & \dfrac{\partial}{\partial y} f_1(p_k, q_k) \\ \dfrac{\partial}{\partial x} f_2(p_k, q_k) & \dfrac{\partial}{\partial y} f_2(p_k, q_k) \end{bmatrix}$$

Paso 3. Calculamos $\Delta\boldsymbol{P}$ resolviendo el sistema lineal

$$\boldsymbol{J}(\boldsymbol{P}_k)\Delta\boldsymbol{P} = -\boldsymbol{F}(\boldsymbol{P}_k)$$

Paso 4. Calculamos el siguiente punto

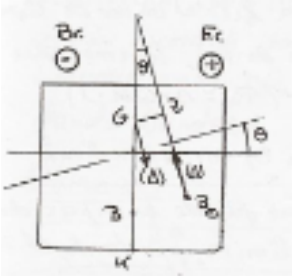
$$\boldsymbol{P}_{k+1} = \boldsymbol{P}_k + \Delta\boldsymbol{P}.$$

Y se repite el proceso.

Ejemplo de aplicación en el ámbito Naval.

Para ejemplificar algunos de los métodos desarrollados anteriormente y poder aplicarlo a un problema real de la ingeniería naval utilizaremos los métodos descritos para hallar las raíces de la curva de brazos adrizantes de un buque, obteniendo así las características más importantes de dicha curva.

Para comenzar, repasaremos el concepto de **curva de brazos adrizantes**.



Cuando por efecto de una acción escorante, un buque se separa de su posición de equilibrio, el c.d.c. modifica su posición ($B \rightarrow B_\theta$). Esto hace que deje de estar alineado con el c.d.g. ,G, apareciendo así un par, que debe tratar de que el buque regrese a su antigua posición de equilibrio. A este par se le conoce como par adrizante (P_a)

Figura 2.2

Para una determinada condición de navegación ($\Delta, KG\ y\ trimado\ dados$), el valor del par adrizante dependerá de las distintas posiciones que vaya ocupando el c.d.c.(B_θ), y por tanto del ángulo de escora.

El diagrama estático no es más que la representación gráfica de la función $P_a = f(\theta)$. Ahora bien, si tenemos en cuenta que el par adrizante es igual a la fuerza (desplazamiento,Δ) por la mínima distancia (GZ).

$$P_a = \Delta \cdot GZ$$

Y que una vez definida la condición de navegación, el desplazamiento Δ, permanece constante, esta curva será la misma si solamente representamos la función $GZ = f(\theta)$.

En el caso de que un buque tuviese una altura metacéntrica negativa, pero que aún conservase alguna capacidad adrizante, la forma de la curva de brazos adrizantes sería la siguiente:

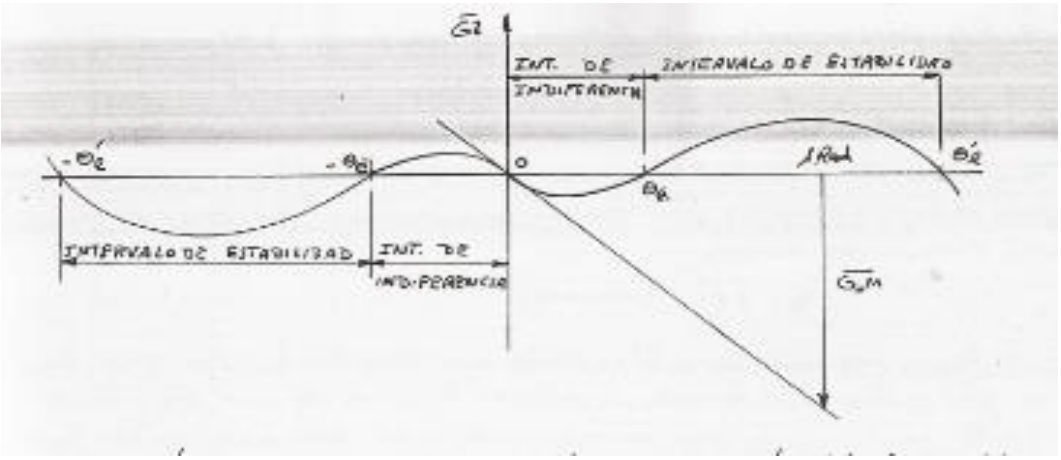


Figura 2.3

En este caso, aunque en el origen tenemos una situación de equilibrio ($GZ = 0$), al ser $G_0M < 0$ el equilibrio es inestable, y el buque acabará escorándose hasta un ángulo θ_e(ángulo que aproximaremos por los diversos métodos iterativos), en el cual alcanza una nueva posición de equilibrio $GZ = 0$.

El método tradicional de obtención de la curva $GZ - \theta$ es a partir de la familia de curvas auxiliares KN. Así, teniendo en cuenta además el efecto por superficies libres:

$$GZ_\theta = KN_\theta - KG * sen(\theta) - \frac{\sum msl}{\Delta}$$

En nuestro ejemplo, disponemos de los siguientes datos del buque:

De un buque pesquero de madera de $L_{pp} = 23{,}400m$ y $B = 6{,}800\ m$ sabemos que para $T = 2m \rightarrow \Delta = 161\ (t)$ y que la curva KN sigue la siguiente función

$$KN = 3{,}334 * \theta(rad) = 0{,}0582 * \theta(º)$$

Suponiendo un $KG = 2{,}450\ m$ y que las superficies libres producen una disminución de 0,01 m de la altura metacéntrica, tendremos la siguiente ecuación del brazo adrizante.

$$GZ_\theta = 0{,}0582\theta - 2{,}450 * sen(\theta) - 0{,}01\ (m)$$

Y suponemos que el ángulo de equilibrio θ_e que queremos hallar se encuentra en un intervalo de $[1, 4]º$.

Pues bien resolveremos este problema aplicando los métodos de aproximación de bisección y regula falsi o Lagrange, para poder apreciar las diferencias en los resultados obtenidos.

Comenzamos aplicando el método de bisección, para ello abrimos la interfaz gráfica e introducimos los datos en esta.

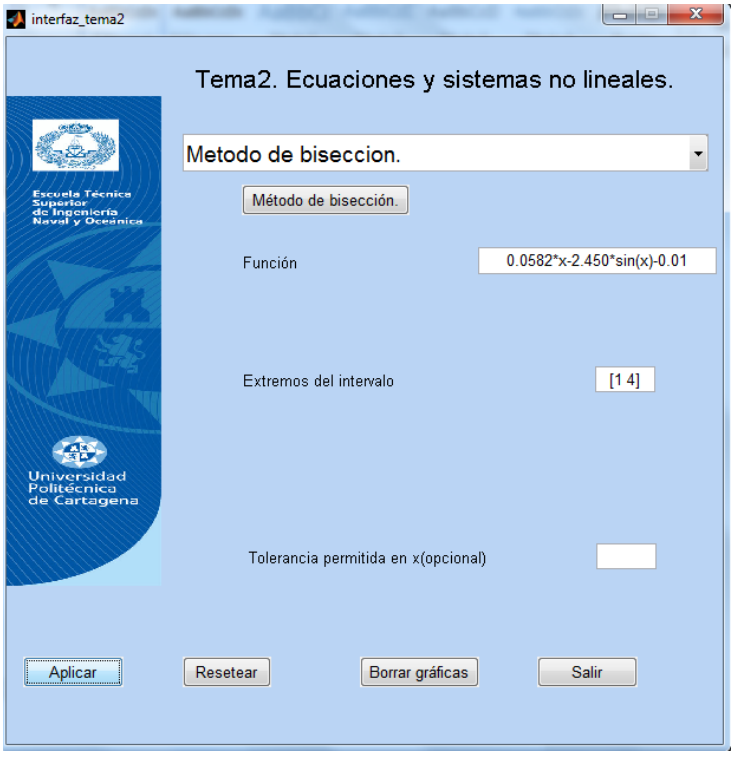


Figura 2.4

Hacemos doble click en el botón aplicar y nos aparecerá la siguiente ventana emergente con los resultados obtenidos;

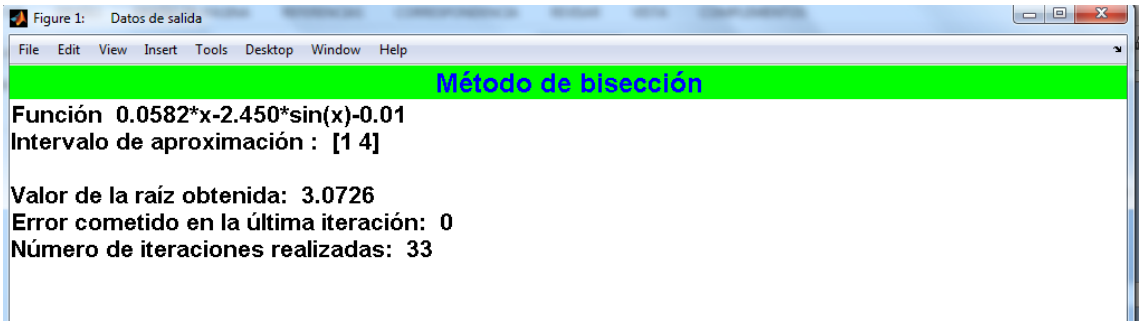


Figura 2.5

Como podemos ver, nuestro nuevo ángulo de equilibrio se encontrará en $3{,}0726º$ exactamente, ya que el error cometido en la última iteración el error cometido es nulo. Sin embargo, para ello el programa ha necesitado realizar 33 iteraciones, con el coste computacional que ello conlleva.

Probamos ahora a resolver el problema mediante el método de Lagrange. Para ello, comenzamos introduciendo los datos en nuestra interfaz gráfica.

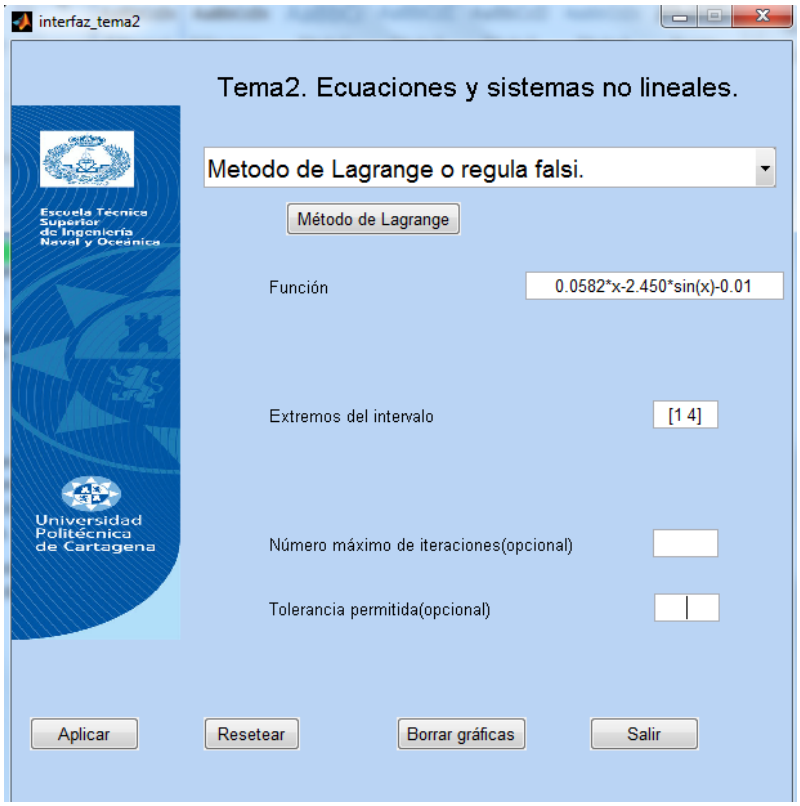


Figura 2.6

Al hacer doble click en el botón aplicar, aparecerá una ventana emergente con los resultados obtenidos;

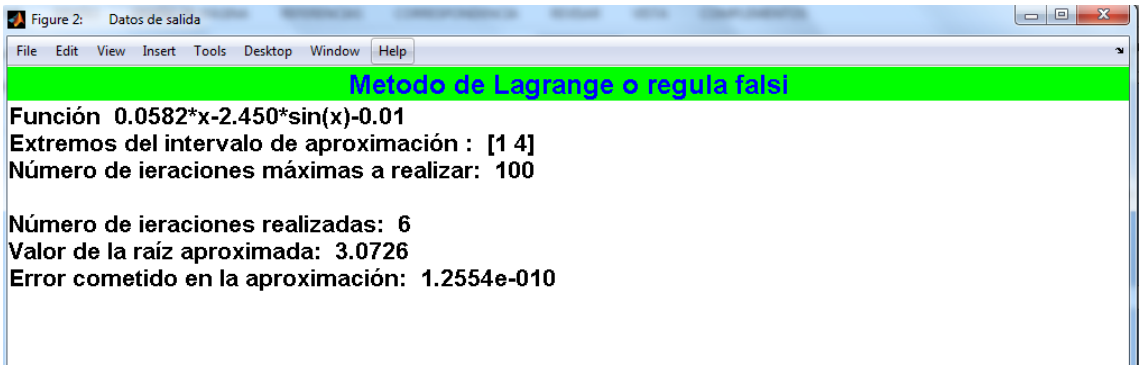


Figura 2.7

Como podemos observar, obtenemos el mismo resultado que en caso anterior, nuestro nuevo ángulo de equilibrio lo encontramos en $3{,}0726º$, con un error prácticamente también nulo. Sin embargo, este método tan sólo ha requerido de 6 iteraciones para hallar la raíz aproximada, siendo pues su orden de convergencia mayor que el del método de bisección, tal y como enunciamos y demostramos en teoría.

Por tanto queda demostrado que con métodos de convergencia mayor podemos obtener los mismos resultados en menor número de iteraciones, ahorrándonos un coste computacional importante.

Ayuda a la interfaz gráfica.

En éste último apartado desarrollaremos una explicación paso a paso de cómo utilizar la interfaz gráfica que ha sido implementada en el programa de análisis matemático Matlab para una mayor comprensión de la teoría descrita mediante ejemplos prácticos.

Para comenzar debemos iniciar el programa Matlab y situar la carpeta que contiene la interfaz gráfica, así como los demás programas a utilizar en éste capítulo, en la carpeta actual de Matlab denominada "Current folder" y hacer doble click sobre la carpeta "Interfaz gráfica" para que sea ésta a la que el programa recurra cuando realicemos una orden en la ventana de comandos.

Para comenzar introducimos la orden " guide" en la ventana de comandos y en la ventana emergente seleccionamos el fichero de nominado" Tema2_Ecuaciones No Lineales".

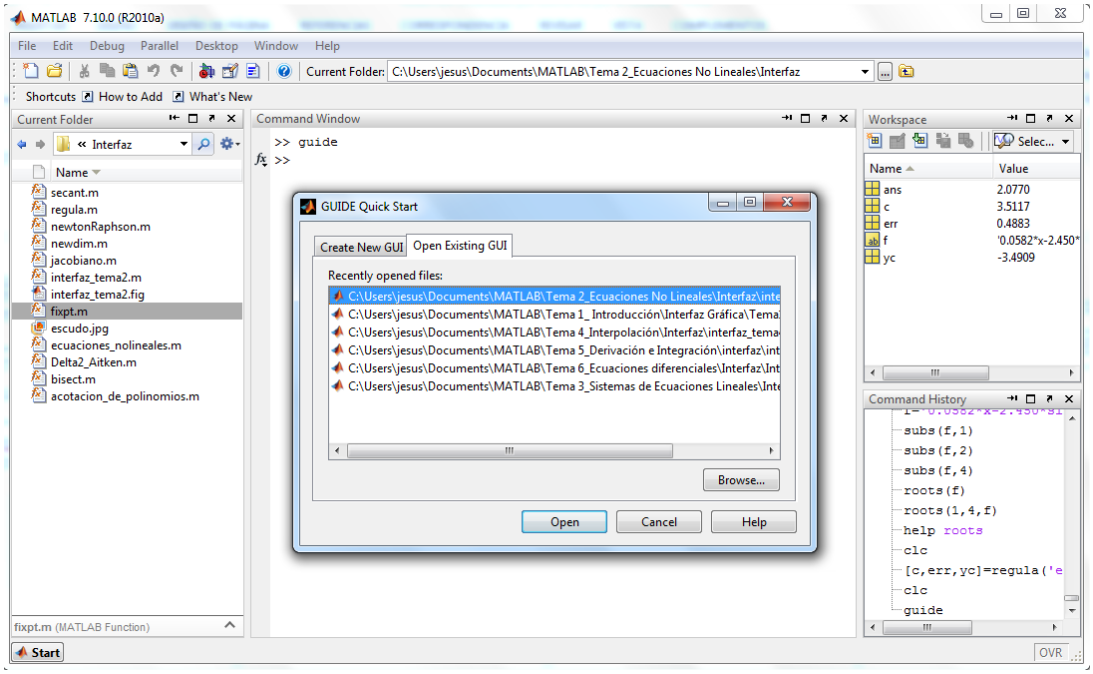


Figura 2.8

Al abrir dicho fichero aparecerá la interfaz de diseño de la propia interfaz gráfica, donde deberemos hacer doble click en el botón verde de reproducir o "play" situado en la zona superior.

Al hacerlo, aparecerá nuestra interfaz gráfica, inicialmente con la lista de métodos que han sido implementados y donde podremos seleccionar el método que queremos reproducir.

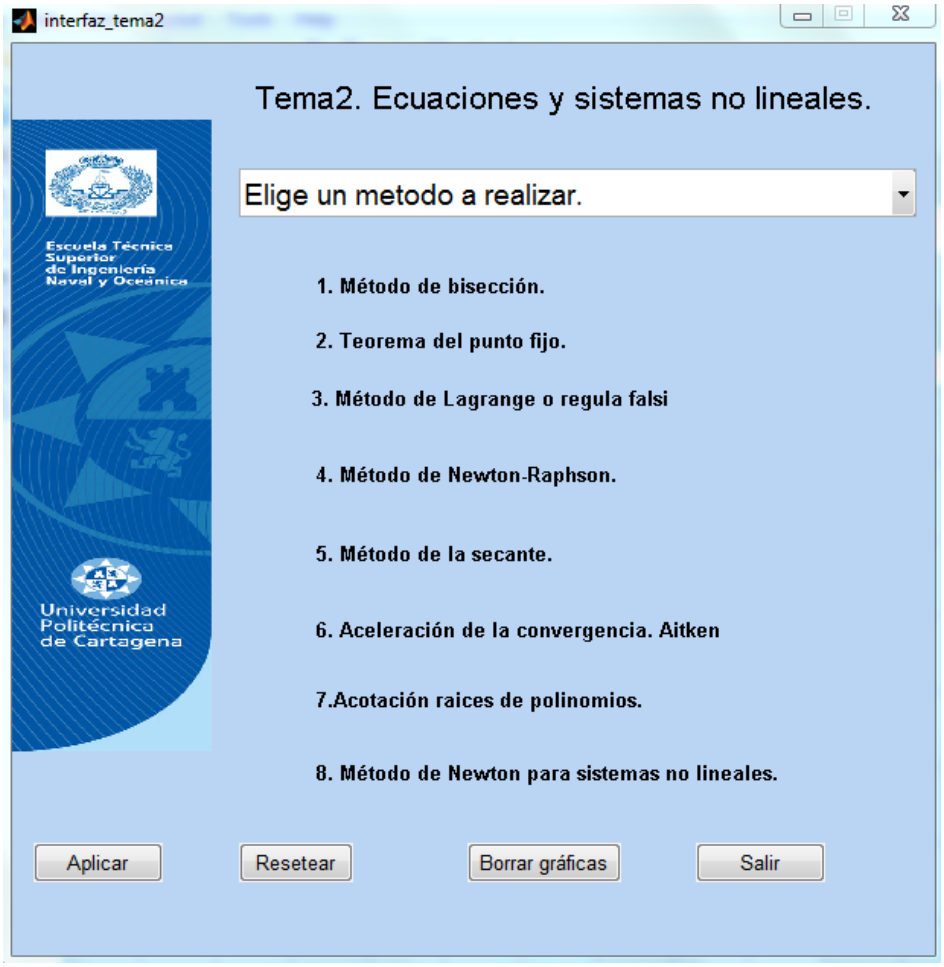


Figura 2.9

En este punto, pulsando en donde dice “Elige un método a realizar” nos aparece automáticamente una lista de los métodos de aproximación de raíces para sistemas y ecuaciones no lineales, donde elegiremos el que deseemos llevar a cabo.

1. Método de bisección.

Seleccionamos el método en la pantalla de inicio. Para comenzar, si presionamos el botón denominado " Método de bisección" nos aparece en pantalla un pdf con la teoría y ejemplos necesarios para comprender el método.

Probaremos este método con la función $f(x) = e^x - 3x$ y hallaremos la raíz comprendida en el intervalo $[0,1]$. Para ello introducimos los datos en la interfaz gráfica del siguiente modo,

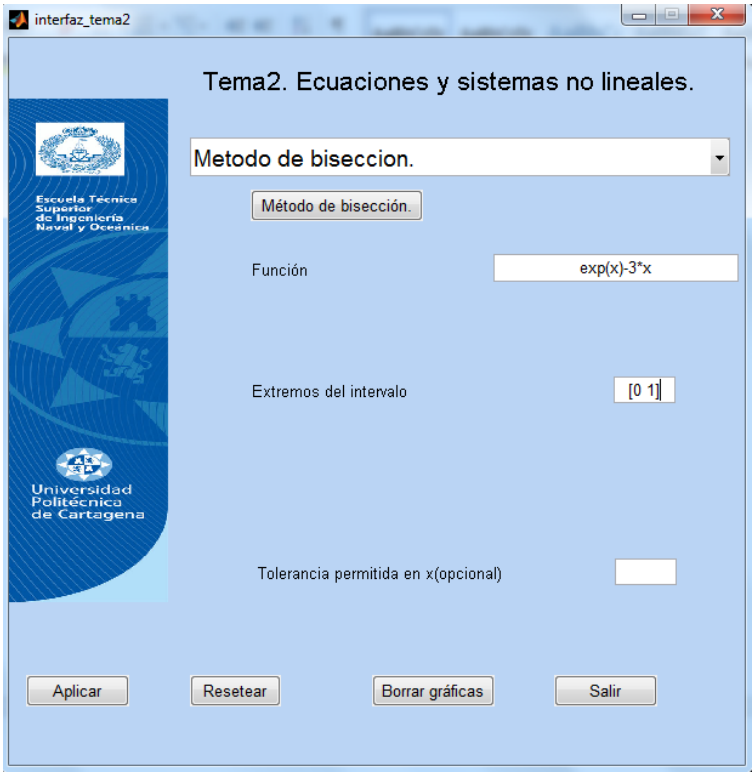


Figura 2.10

Al hacer click en el botón aceptar, nos aparece una ventana emergente con los resultados obtenidos,

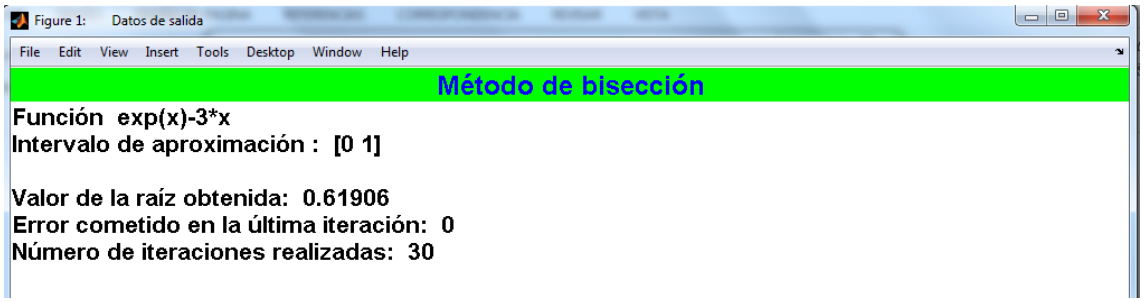


Figura 2.11

Como podemos observar, obtenemos que la raíz se encuentra en el punto $0{,}61906$ siendo el error cometido nulo. Sin embargo, el número de iteraciones realizadas es alto, con el coste computacional que ello conlleva.

2. Teorema del punto fijo.

Seleccionamos el método en la pantalla de inicio. Para comenzar, si presionamos el botón denominado " Teorema del punto fijo" nos aparece en pantalla un pdf con la teoría y ejemplos necesarios para comprender el método.

Probaremos este método con la función $f(x) = 1 + x - \frac{x^2}{4}$ comenzando las iteraciones en el punto $x = 0$. Para ello, introducimos los datos en la interfaz,

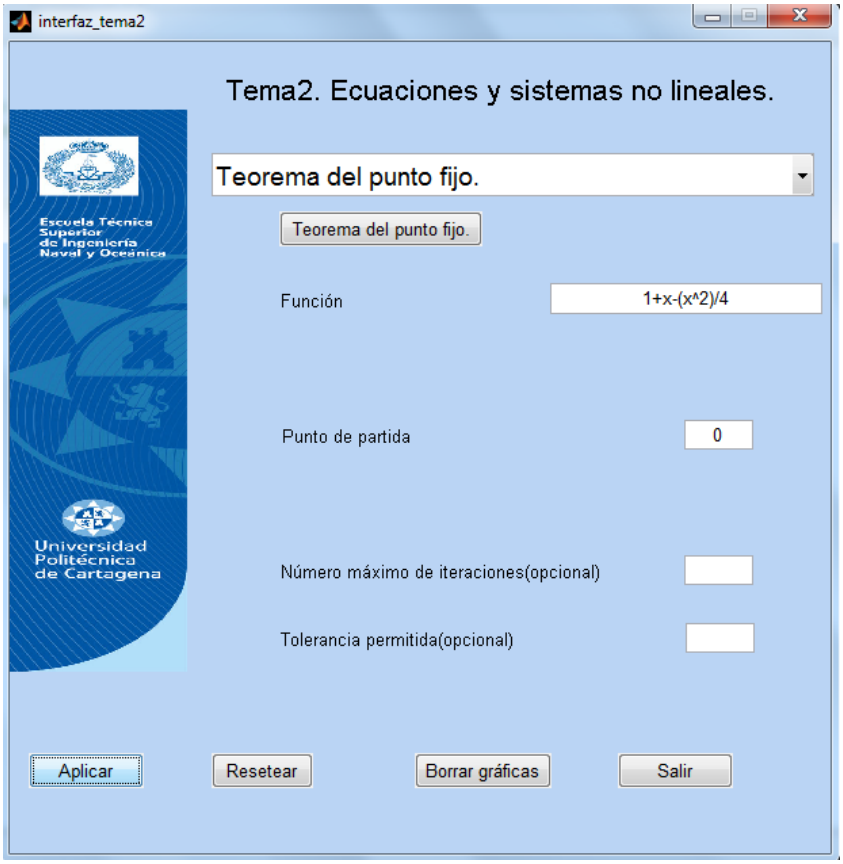


Figura 2.12

Al hacer click en el botón aplicar aparece una ventana emergente con los resultados obtenidos,

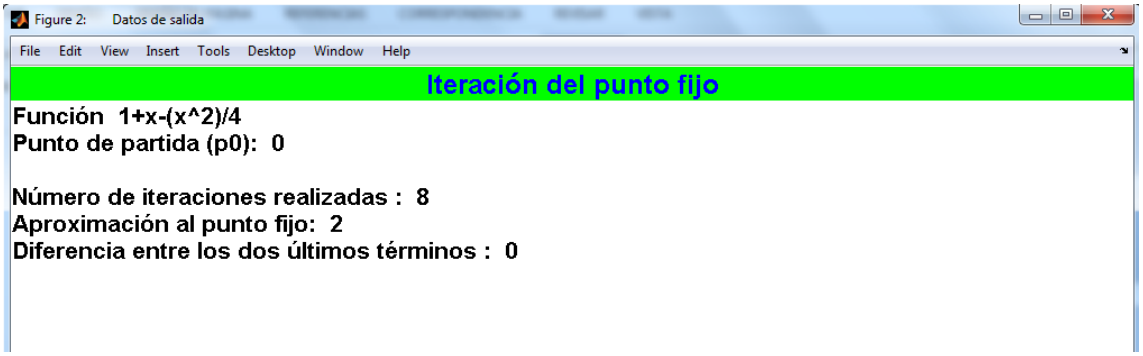


Figura 2.13

Como podemos observar, obtenemos que la raíz se encuentra en el punto $x = 2$, lo que es fácilmente comprobable. Por otro lado, el programa ha necesitado un total de 8 iteraciones para hallar dicha raíz, si hubiéramos partido de un punto más cercano a la raíz como $x = 1$, el número de iteraciones se vería reducido.

3. Método de Lagrange o regula falsi.

Seleccionamos el método en la pantalla de inicio. Para comenzar, si presionamos el botón denominado " Método de Lagrange" nos aparece en pantalla un pdf con la teoría y ejemplos necesarios para comprender el método.

Probaremos este método con la función $f(x) = e^x - 3x^2$ y obtendremos la raíz comprendida en el intervalo $[3\ 4]$ y deseamos que realice un máximo de 50 iteraciones. Para ello introducimos los datos en la interfaz,

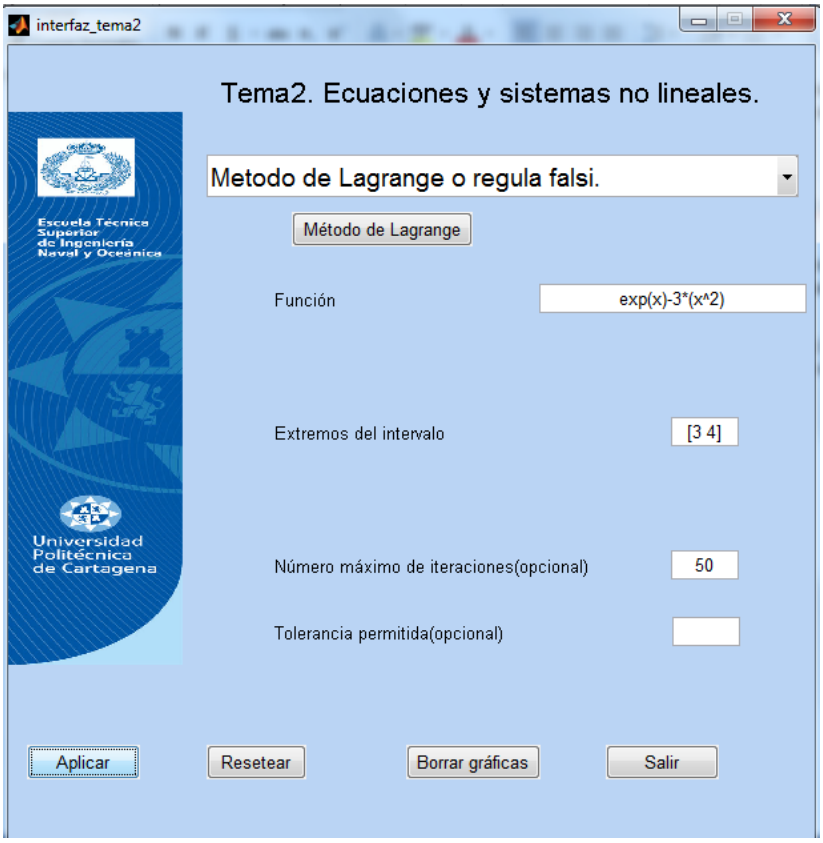


Figura 2.14

Al hacer click en el botón aplicar aparece una ventana emergente con los resultados obtenidos.

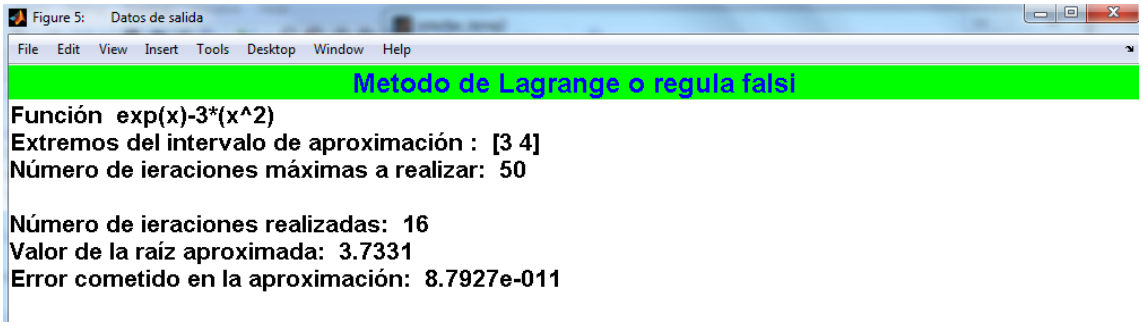


Figura 2.15

Como podemos observar, la raíz se encuentra en el punto $x = 3{,}7331$. También podemos ver que el número máximo de iteraciones no se ha superado, en este caso 50, habiendo tenido que realizar únicamente 16 iteraciones.

4. Método de Newton-Raphson.

Seleccionamos el método en la pantalla de inicio. Para comenzar, si presionamos el botón denominado " Método de Newton-Raphson" nos aparece en pantalla un pdf con la teoría y ejemplos necesarios para comprender el método.

Probaremos este método con la función $f(x) = \cos(x) - x$ en el intervalo $\left[0.5\frac{\pi}{4}\right]$, comenzando en el punto $x = 1$ y consideramos suficiente una aproximación a la raíz igual a $0{,}01$. Para ello introducimos los datos en la interfaz,

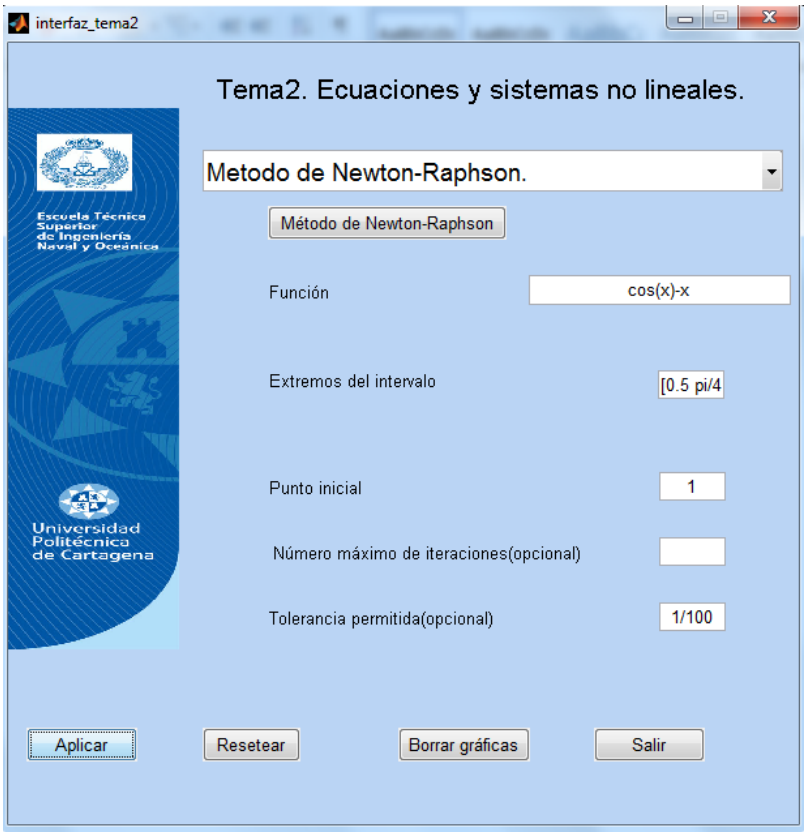


Figura 2.16

Al hacer click en el botón aplicar aparece una ventana emergente con los resultados obtenidos,

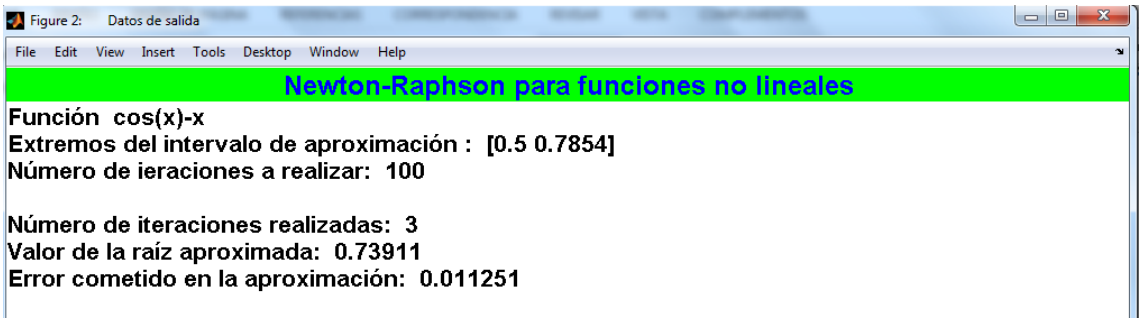


Figura 2.17

Como podemos observar, la raíz obtenida se encuentra en el punto $x = 0.73911$. Además, el programa ha necesitado tan sólo un total de 3 iteraciones para encontrarla con el error que pedido.

5. Método de la secante.

Seleccionamos el método en la pantalla de inicio. Para comenzar, si presionamos el botón denominado " Método de Newton-Raphson" nos aparece en pantalla un pdf con la teoría y ejemplos necesarios para comprender el método.

Probamos este método con la misma función que en el caso anterior, de este modo podremos observar las diferencias en los resultados obtenidos. En este caso, en lugar de introducir el intervalo en que queremos que encuentre la raíz, se introduce el primer y segundo punto por los que comenzar las iteraciones. Además le pedimos el mismo grado de aproximación que en el caso anterior para así poder comparar resultados.

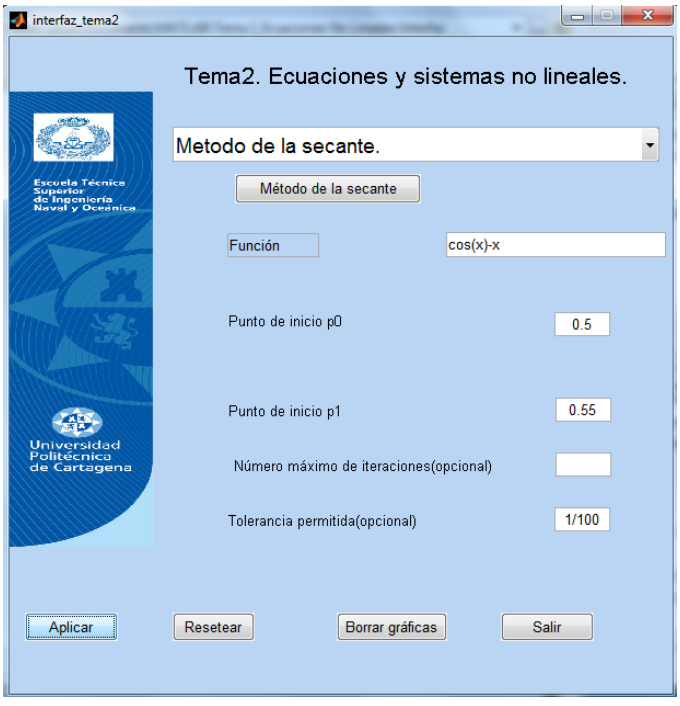


Figura 2.18

Al hacer click en el botón aplicar aparece una ventana emergente con los resultados obtenidos.

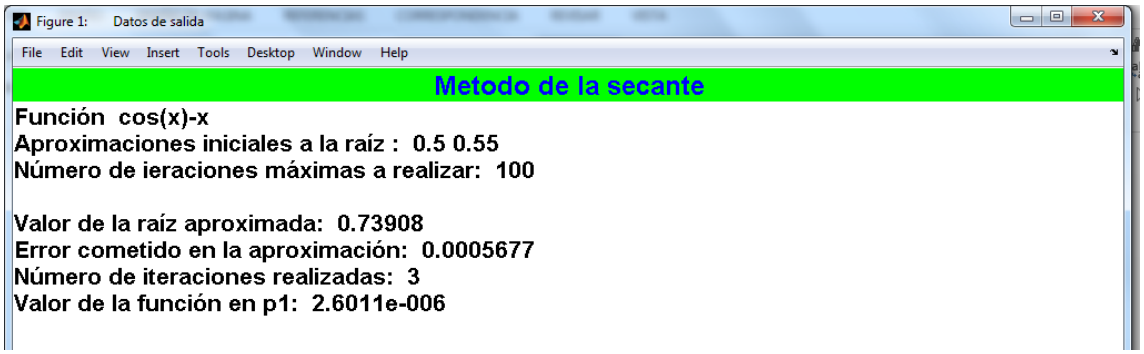


Figura 2.19

Como podemos observar, el resultado difiere ligeramente en la cuarta cifra decimal en comparación con el resultado anterior, en este caso se ha obtenido la raíz en el punto

$x = 0{,}73908$. Por otro lado, en este caso también ha necesitado de 3 iteraciones para hallar la solución cometiendo, sin embargo, un error considerablemente menor, lo cual se contradice con lo demostrado en teoría en la cual se afirma que el método de Newton-Raphson posee un orden de convergencia $R = 2$, mientras que el método de la secante posee un orden

$R = 1.618033989$. Por lo tanto suponemos que la mejor aproximación obtenida por el método de la secante se debe a la acumulación de errores que se hayan podido cometer en el método de Newton-Raphson a la hora del cálculo de derivadas sucesivas.

6. Método de aceleración de la convergencia de Aitken.

Seleccionamos el método en la pantalla de inicio. Para comenzar, si presionamos el botón denominado “Aceleración de la convergencia. Aitken” nos aparece en pantalla un pdf con la teoría y ejemplos necesarios para comprender el método.

Para probar este método hallamos las raíces de la función $f(x) = x^2 - x$ partiendo del punto $x = 0.5$, permitiendo un número máximo de diez aceleraciones. Para ello introducimos los datos en la interfaz,

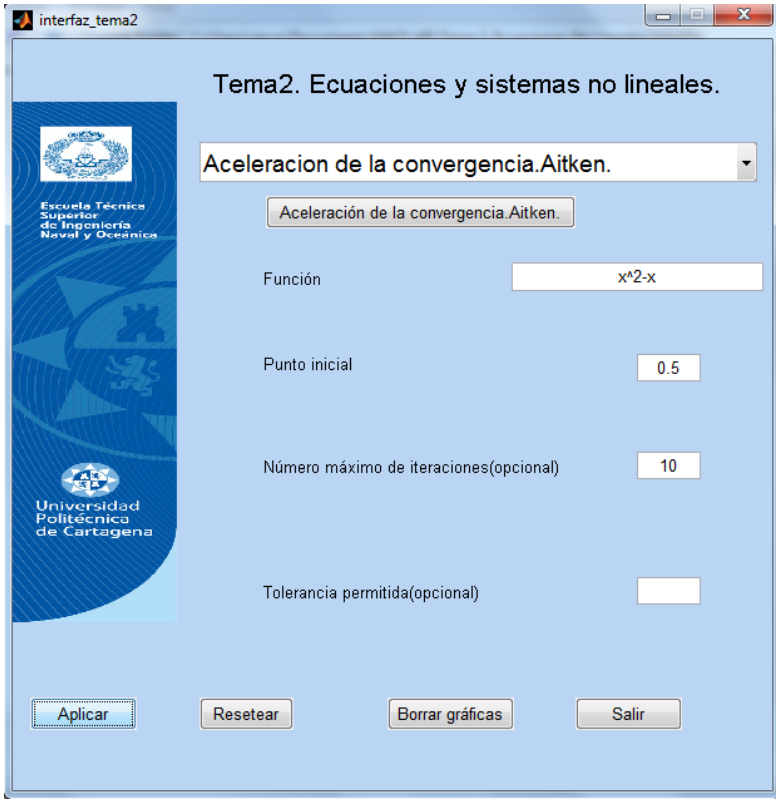


Figura 2.20

Al hacer click en el botón aplicar aparece una ventana emergente con los resultados obtenidos,

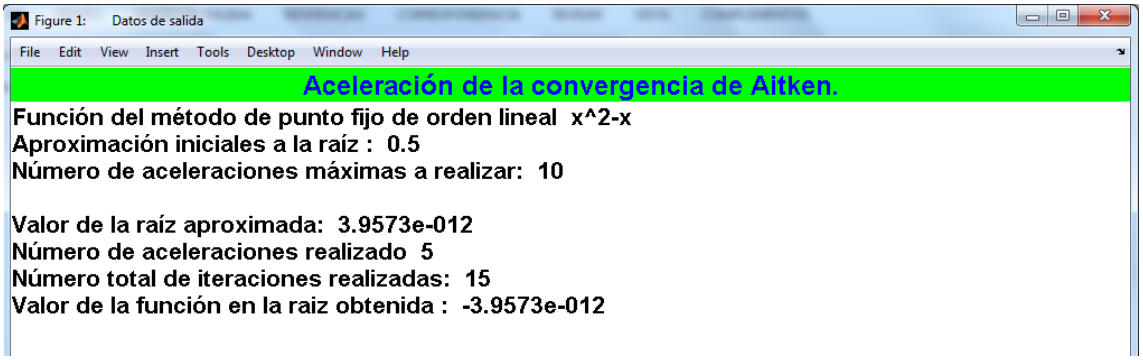


Figura 2.21

Como podemos observar, con tan solo cinco aceleraciones hemos sido capaces de hallar una raíz con un margen de error casi nulo.

7. Acotación de raíces de una ecuación polinómica

Seleccionamos el método en la pantalla de inicio. Para comenzar, si presionamos el botón denominado “Acotación raíces de un polinomio” nos aparece en pantalla un pdf con la teoría y ejemplos necesarios para comprender el método.

Para probar este método utilizaremos la ecuación polinómica

$$p(x) = x^7 - 2x^6 + x^5 - x^4 + 2x^3 - 3x^2 + 7x - 1 = 0$$

Además, suponemos que la cota máxima de las raíces estará como mucho en $x = 10$.

Para ello introducimos los datos en la interfaz.

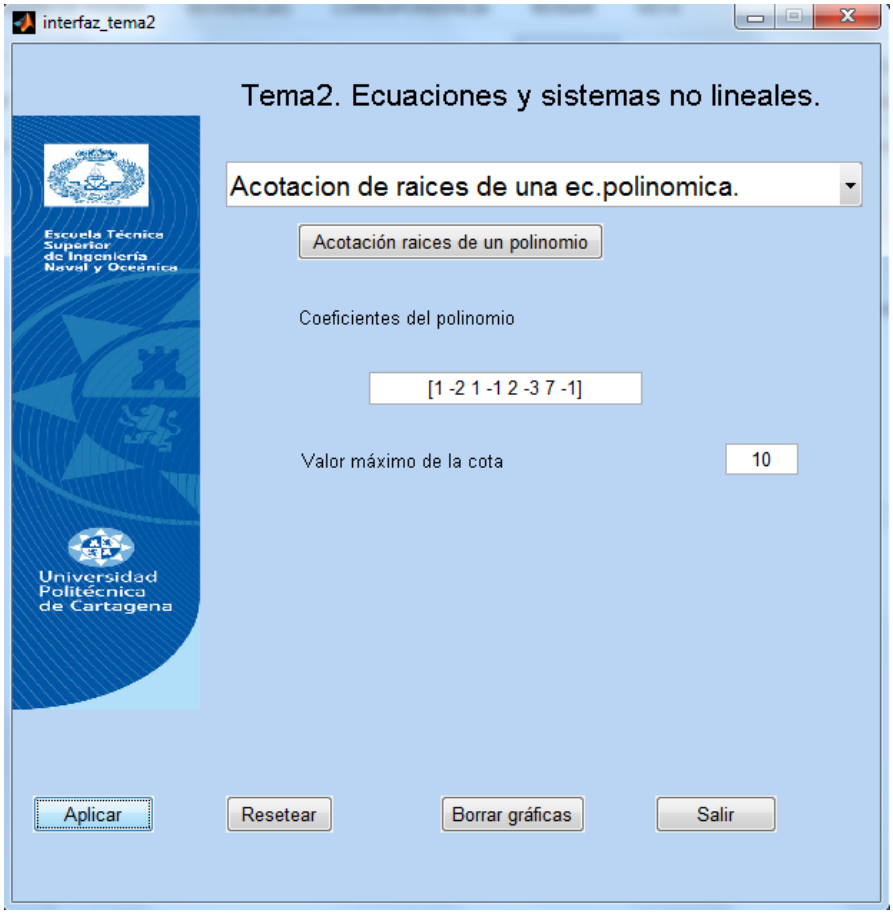


Figura 2.22

Al hacer click en el botón aplicar aparece una ventana emergente con los resultados obtenidos.

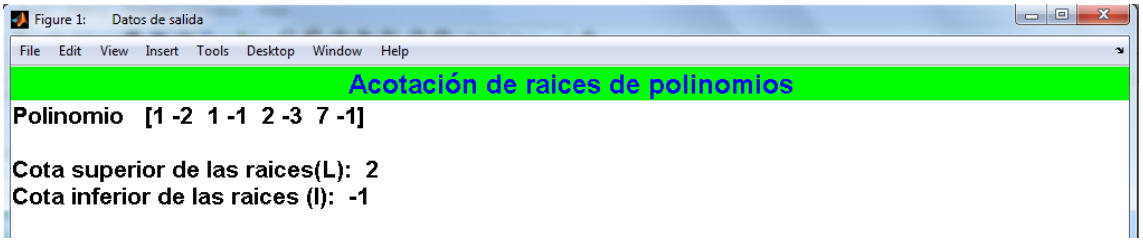


Figura 2.23

Por lo que las raíces de la ecuación polinómica se encontrarán en el intervalo $x \in [-1\ 2]$

8. Método de Newton para sistemas no lineales.

Seleccionamos el método en la pantalla de inicio. Para comenzar, si presionamos el botón denominado “Newton para sistemas no lineales” nos aparece en pantalla un pdf con la teoría y ejemplos necesarios para comprender el método.

Para probar este método trataremos de resolver el siguiente sistema de ecuaciones no lineales formado por dos ecuaciones y dos incógnitas, aunque podría estar formado por tantas ecuaciones e incógnitas como deseáramos;

$$\begin{cases} e^x - y \\ x + y - 3 \end{cases}$$

Partiendo del punto de inicio $(x_0, y_0) = (0.5\ 0.5)$. Además admitimos un número máximo de 20 iteraciones. Para ello introducimos los datos en la interfaz gráfica. Para introducir tanto las ecuaciones como el punto de inicio de las iteraciones, debemos pulsar el botón denominado “Crear función de sistemas no lineales y punto inicial”, entonces el programa nos remite a un archivo en el que podremos introducir dichos datos, en donde además se encuentra explicado con detalle el modo de introducirlo, incluyendo ejemplos.

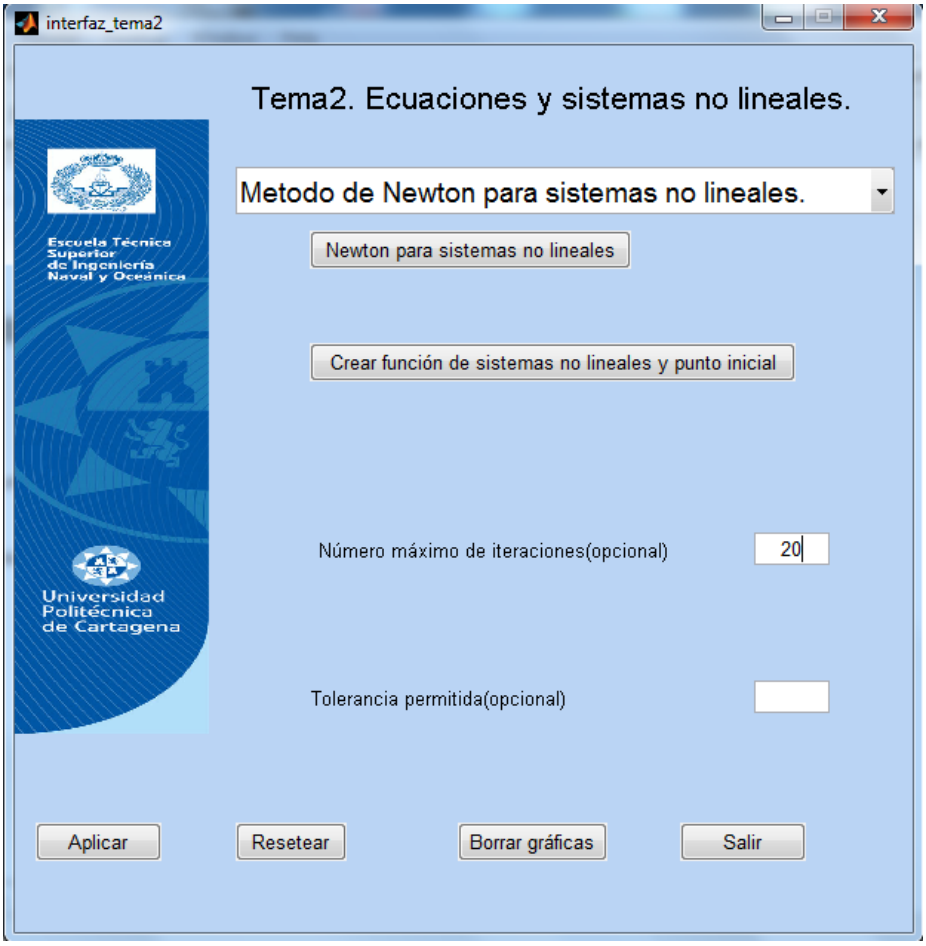


Figura 2.24

Cuando ya hemos terminado de introducir los datos del sistema y el punto de inicio en el archivo aparte, y los guardamos, al hacer click en el botón aplicar aparece un archivo de texto con los resultados obtenidos;

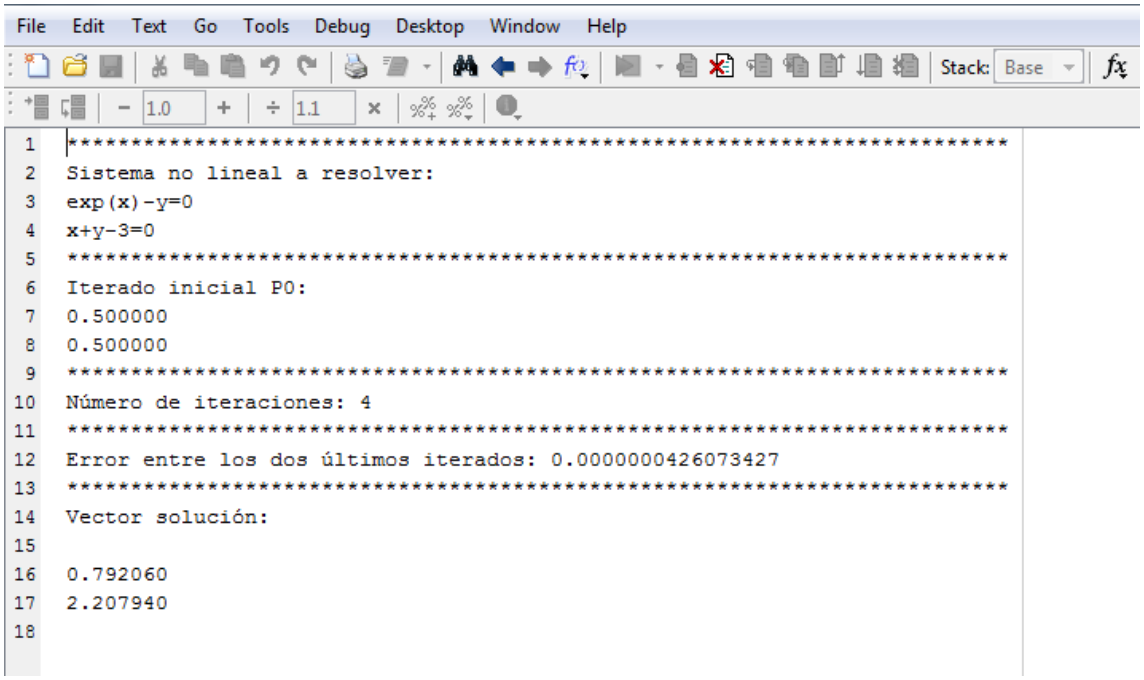


Figura 2.25

Como podemos observar, se han realizado un total de cuatro iteraciones, obteniendo como resultado el punto $(x, y) = (0.792060\,, 2.207940)$ como raíz del sistema de ecuaciones no lineal.

Capítulo 3.Resolución numérica de sistemas lineales.

Introducción. Normas vectoriales y matriciales.

Los métodos de resolución de sistemas lineales $Ax = b$ los podemos clasificar en directos (que persiguen obtener la solución exacta en un número finito de pasos, como por ejemplo los métodos de Kramer, Gauss, LU, Cholesky, etc) e iteraivos (en los que se parte de unos valores aproximados iniciales y a partir de ellos se van obteniendo valores cada vez más aproximados de la solución $x = A^{-1}b$, ejemplos de tales métodos son los de Jacobi, Gauss-Seidel,etc).

Para unos y otros se define la estabilidad numérica como la sensibilidad del método utilizado respecto a pequeñas perturbaciones o errores en los datos. Seguidamente damos las definiciones de normas vectoriales y matriciales que nos permitirán dar resultados sobre convergencia y acotación de errores en unos y otros.

Definición 1.

Sea $(E\ , k, +, \cdot)$ un espacio vectorial. Una norma en E es cualquier aplicación:

$$||\cdot||: E \to k$$

Que verifique $\forall \lambda \in k \quad y\ \forall z, w \in E$;

1. $\big||z|\big| \geq 0 \quad y \quad \big||z|\big| = 0 <=> z = 0$
2. $\big||\lambda|\big| = |\lambda|\ ||z||$
3. $\big||z + w|\big| \leq \big||z|\big| + ||w||$ (Desigualdad triangular).

Tres ejemplos clásicos de norma en $E = C^n$.

4. Norma1. $||z||_1 = |z_1| + \cdots |z_n|$
5. Norma2 o euclídea.
   $||z||_2 = \sqrt{|z_1|^2 + \cdots |z_n|^2}$
6. Norma∞ o norma del supremo.
   $||z||_\infty = \max\{|z_1|, \ldots, |z_n|\}$

Las normas 1 y 2 son casos particulares de la norma p, definida $\forall p \in N$ como

$$||z||_p = (|z_1|^p + \cdots |z_n|^p)^{1/p} \qquad (1)$$

Definición2.

Se dice que dos normas $||\cdot||_*, ||\cdot||_\blacksquare$ en E son equivalentes si $\exists\ \alpha, \beta > 0$ tal que

$$\forall u\ \in E,\ \ \alpha||u||_* \leq ||u||_\blacksquare \leq \beta||u||_* \qquad (2)$$

Todo producto escalar $<*,*>$ en E induce una norma en E, definida como

$$\big||z|\big| = \sqrt{< z, z >} \qquad (3)$$

Norma matricial inducida por normas vectoriales.

Sean $||\cdot||_*, ||\cdot||_\blacksquare$ dos normas en C^m, C^n se denomina norma matricial en $||\cdot||$ en C^{mxn} inducida por dichas normas vectoriales a

$$||\cdot||: C^{mxn} \to R$$

$$A \to \left|\left|A\right|\right| = \sup_{x\neq 0} \frac{||Ax||_*}{||x||_\blacksquare} \qquad (4)$$

Verifica las propiedades de la norma:

1. $\left(\forall\, x\ \in C^n, \frac{||A||_*}{||A||_\blacksquare} \geq 0\right) \to ||A|| \geq 0.$

   y $\left|\left|A\right|\right| = 0 \leftrightarrow\ \forall\, x \in C^n, \left|\left|Ax\right|\right|_* = 0 \leftrightarrow\ \ \forall x \in C^n, Ax = 0\ \leftrightarrow A = [0]$

2. $\left|\left|\lambda A\right|\right| = sup_{x\neq 0} \frac{||\lambda A||_*}{||x||_\blacksquare} = |\lambda| \sup_{x\neq 0} \frac{||Ax||_*}{||x||_\blacksquare} = |\lambda|||A||$

3. $\left|\left|A + B\right|\right| = \sup_{x\neq 0} \frac{||Ax+Bx||_*}{||x||_\blacksquare} \leq \sup_{x\neq 0} \left[\frac{||Ax||_*}{||x||_\blacksquare} + \frac{||Bx||_*}{||x||_\blacksquare}\right] \leq \sup_{x\neq 0} \frac{||Ax||_*}{||x||_\blacksquare} + \sup_{x\neq 0} \frac{||Bx||_*}{||x||_\blacksquare} \leq \left|\left|A\right|\right| + ||B||.$

Definición3. El coeficiente $\frac{||Ax||}{||x||}$ alcanza su supremo en $C^n \setminus \{0\}$. Permite sustituir el término "supremo" por "máximo".

$$\left|\left|A\right|\right| = \max_{x\neq 0} \frac{\left|\left|Ax\right|\right|}{||x||} = \max_{x=1} \left|\left|Ax\right|\right|. \qquad (5)$$

Si $||\cdot||$ es una norma en C^{nxm} inducida por normas vectoriales, entonces

$$\forall\, A, B\ \in C^{nxm},\ \left|\left|A\ B\right|\right| \leq \left|\left|A\right|\right|\ ||B|| \qquad (6)$$

<u>Algunos ejemplos.</u>

Norma 1.

Si las normas $||\cdot||_*, ||\cdot||_\blacksquare$ son las normas 1 en $C^n\ y\ C^m$,

$$\left|\left|A_1\right|\right| = \max_{||x_1||=1} ||Ax||_1 = \max_{1<k\leq n} \left\{\sum_{j=1}^{m} |a_{jk}|\right\} \qquad (7)$$

O sea, es el máximo de las normas subuno de los vectores columna de la matriz dada.

Norma 2 o norma espectral.

Si las normas $||\cdot||_*, ||\cdot||_\blacksquare$ son las normas 2 en $C^n\ y\ C^m$,

$$\left|\left|A_2\right|\right| = \sqrt{\rho(A^h A)} \qquad (8)$$

Siendo ρ el radio espectral.

$\rho(M) = \max_{1\leq i\leq n}\{|\lambda_i|\}$. Siendo λ_i los valores propios de A.

Recordemos que A^hA es la matriz hermítica, y por tanto diagonalizable unitariamente con autovalores reales $\lambda_1 \leq \cdots \leq \lambda_n$. Siendo éstos no negativos.

*Una matriz Hermitiana o Hermítica es una matriz cuadrada de elementos complejos que tiene la característica de ser igual igual a su propia transpuesta conjugada.

Una matriz transpuesta conjugada o adjunta de una matriz A es una matriz $A^+ \, o \, A^h$, obtenida mediante su transpuesta y después de su conjugada compleja.

Una matriz conjugada es el resultado de la sustitución de los elementos de una matriz A por sus conjugados, es decir, la parte imaginaria de los elementos de la matriz cambia de signo.

Norma ∞.

$$||A||_\infty = max_i \sum_{j=1}^{n} |a_{ij}| \qquad (9)$$

O sea, el máximo de las normas subuno de los vectores fila de la matriz dada.

Número de condición de una matriz.

Sea $A \in C^{nxm}$, con $rg(A) = n \leq m$. Se define el número de condición de A asociado a una norma $||\cdot||$ como

$$k(A) = \frac{\max\limits_{||x||_1=1} ||Ax||}{\min\limits_{||x||=1} ||Ax||} = \frac{M}{m} = \big|\big|A\big|\big| \, ||A^{-1}|| \qquad (10)$$

Cuando A no es inversible se define $k(A) = \infty$. Es fácil probar que siempre se cumple $k(A) \geq 1$, cuando $k(A) = 1$ se dice que A está perfectamente condicionada.

Perturbación del término independiente b.

Se considera $Ax = b$, con $A \in C^{mxn}$ y $rg(A) = nxm$. Supongamos que tiene la solución única $x_0 \neq 0$. Se desea estudiar la variación del vector $x_0$ante variaciones del vector b.

$$Ax = b + \Delta b \qquad (11)$$

Asumimos que tenemos la solución única $x_0 + \Delta x_0$. Se desea estimar $||\Delta x_0||$.

$$A(\Delta x_0) = \Delta b \begin{cases} Ax_0 = b \\ A(x_0 + \Delta x_0) = b + \Delta b \end{cases} \qquad (12)$$

Definimos $M = \max\limits_{||x||=1} ||Ax||$ y $m = \min\limits_{||x||=1} ||Ax||$

$$m = \min_{||x||\neq 0} \frac{||Ax||}{||x||} \leq \frac{||A\,\Delta x_0||}{||\Delta x_0||} = \frac{||\Delta b||}{||x_0||} \rightarrow ||\Delta x_0|| \leq \frac{||\Delta b||}{m}$$

$$M = \max_{||x||\neq 0} \frac{||Ax||}{||x||} \geq \frac{||A\,\Delta x_0||}{||\Delta x_0||} = \frac{||b||}{||x_0||} \rightarrow \frac{1}{||x_0||} \leq \frac{M}{||b||}$$

Multiplicando ambas ecuaciones,

$$\frac{||\Delta x_0||}{||x_0||} \leq \frac{M}{m} \frac{||\Delta b||}{||b||} = c(A) \frac{||\Delta b||}{||b||} \qquad (13)$$

Es decir que el error relativo en x es menor o igual que $k(A)$ veces el error relativo en b.

Si el número de condición es grande se dice que la matriz está mal condicionada y la solución puede ser muy sensible con respecto a pequeños cambios en los términos independientes, en caso contrario se dice que está bien condicionada.

Métodos directos de resolución de sistemas lineales.

En general, dado un sistema $Ax = b$ de m ecuaciones lineales con n incógnitas, si a la matriz ampliada $(A, b)_{m\,x\,(n+1)}$ le efectuamos operaciones elementales de filas (lo cual equivale a multiplicar por matrices regulares de orden m a la izquierda), se obtiene una nueva matriz ampliada $(\bar{A}, \bar{b})_{m\,x(n+1)}$; entonces x_0 es solución del sistema $Ax = b$ si y solo si es solución del sistema $\bar{A}x = \bar{b}$.

Sistemas lineales triangulares.

Desarrollaremos ahora el algoritmo de sustitución regresiva, con el que podremos resolver un sistema de ecuaciones lineales cuya matriz de coeficientes sea triangular superior.

Se dice que una matriz $A = [a_{ij}]$ de orden $N\ x\ N$ es triangular superior cuando sus elementos verifican $a_{ij} = 0$ siempre que $i > j$. Se dice que una matriz $A = [a_{ij}]$ de orden $N \times N$ es triangular inferior si $a_{ij} = 0$ siempre que $i < j$.

Vamos a desarrollar un método para hallar la solución de un sistema de ecuaciones lineales triangular superior. Si A es una matriz triangular superior, entonces se dice que el sistema de ecuaciones $AX = B$ es un sistema triangular superior de ecuaciones lineales, sistema que tiene la siguiente forma:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1N-1}x_{N-1} + a_{1N}x_N &= b_1 \\ a_{22}x_2 + a_{23}x_3 + \cdots + a_{2N-1}x_{N-1} + a_{2N}x_N &= b_2 \\ a_{33}x_3 + \cdots + a_{3N-1}x_{N-1} + a_{3N}x_N &= b_3 \\ \vdots \qquad\qquad & \vdots \\ a_{N-1N-1}x_{N-1} + a_{N-1N}x_N &= b_{N-1} \\ a_{NN}x_N &= b_N. \end{aligned} \qquad (14)$$

Sustitución regresiva. Supongamos que $AX = B$ es un sistema triangular superior como el dado en (14). Si

$$a_{kk} \neq 0 \qquad para\ k = 1,2, \dots, N, \qquad (15)$$

Entonces existe una solución única de (14).

La solución es fácil de hallar. La última ecuación sólo contiene la incógnita x_N, así que empezamos por ésta:

$$x_N = \frac{b_N}{a_{NN}} \qquad (16)$$

Ahora ya conocemos x_N así que podemos usarla en la penúltima ecuación:

$$x_{N-1} = \frac{b_{N-1} - a_{N-1}x_N}{a_{N-1N-1}} \qquad (17)$$

Ahora usamos x_N, x_{N-1} para hallar x_{N-2}:

$$x_{N-2} = \frac{b_{N-2} - a_{N-2N-1}x_{N-1} - a_{N-2}x_N}{a_{N-2N-2}} \qquad (18)$$

Una vez calculados los valores $x_N, x_{N-1}, \dots, x_{k+1}$, el paso general es

$$x_k = \frac{b_k - \sum_{j=k+1}^{N} a_{kj}x_j}{a_{kk}} \quad para\ k = N-1, N-2, \dots, 1 \qquad (19)$$

La unicidad de la solución es fácil de ver. La última ecuación implica que b_N/a_{NN} es el único posible valor de x_N y, por inducción finita, los valores de $x_{N-1}, x_{N-2}, \dots, x_1$ también son únicos.

Si una matriz $A = [a_{ij}]$ de orden $N \times N$ es triangular superior o inferior, entonces

$$\det(A) = a_{11}a_{22} \dots a_{NN} = \prod_{i=1}^{N} a_{ii} \qquad (20)$$

Eliminación gaussiana y pivoteo.

En esta sección desarrollaremos un método para resolver un sistema de ecuaciones lineales general $AX = B$ de N ecuaciones con N incógnitas. El objetivo es construir un sistema triangular superior equivalente $UX = Y$ que podamos resolver utilizando el método de la sustitución regresiva.

Se dice que dos sistemas de orden $N \times N$ son equivalentes cuando tienen el mismo conjunto de soluciones. Los teoremas de álgebra lineal prueban que hay ciertas transformaciones que no cambian el conjunto de soluciones de un sistema de ecuaciones lineales.

Operaciones elementales con filas.

Cualquiera de las siguientes operaciones aplicada a una matriz produce un sistema lineal equivalente.

1. Intercambio: El orden de las filas puede cambiarse.
2. Escalado: Multiplicar una fila por una constante no nula.
3. Sustitución: Una fila puede ser reemplazada por la suma de esa fila más el múltiplo de cualquier otra fila; o sea,

$$fila_r = fila_r - m_{rq} \times fila_q$$

Como se conoce de los cursos previos de álgebra lineal, la forma habitual de usar la sustitución es reemplazar una fila por la diferencia entre esa fila y un múltiplo de otra.

Una forma eficaz de trabajar es almacenar todas las constantes del sistema lineal $AX = B$ en una matriz de orden $N \times (N+1)$ que se obtiene añadiendo a la matriz A una columna, la columna $(N+1) - ésima$, en la que se almacenan los términos de B (es decir, $a_{kN+1} = b_k$). Cada fila de esta matriz, que se llama matriz ampliada del sistema y se denota por $[A|B]$, contiene toda la información necesaria para representar la correspondiente ecuación del sistema lineal:

$$[A|B] = \left[\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1N} & b_1 \\ a_{21} & a_{22} & \dots & a_{2N} & b_2 \\ \vdots & \vdots & & \vdots & \vdots \\ a_{N1} & a_{N2} & \dots & a_{NN} & b_N \end{array}\right] \qquad (21)$$

Un sistema $AX = B$, cuya matriz ampliada viene dada en (21), puede resolverse realizando operaciones elementales con las filas de la matriz ampliada $[A|B]$. Las variables x_k no sirven más que para marcar el sitio de los coeficientes y pueden ser omitidas hasta el final de los cálculos.

Pivotes y multiplicadores.

El elemento a_{qq} de la matriz de los coeficientes en el paso $q + 1$ que se usará en la eliminación de a_{rq}, para $r = q + 1, q + 2, \dots, N$, se llama q-ésimo pivote y a la fila q-ésima se llama fila pivote. Los números $m_{rq} = a_{rq}/a_{qq}$ $(r = q + 1, q + 2, \dots, N)$ por los que se multiplica la fila pivote para restarla de las correspondientes filas posteriores se llaman multiplicadores de la eliminación.

Eliminación de Gauss con sustitución regresiva.

Si A es una matriz invertible de orden $N \times N$, entonces existe un sistema lineal $UX = Y$, equivalente al sistema $AX = B$, en el que U es una matriz triangular superior con elementos diagonales $u_{kk} \neq 0$. Una vez construidos $U\ e\ Y$,se usa el algoritmo de sustitución regresiva para resolver $UX = Y$ y, así, calcular la solución X.

Usaremos la matriz ampliada del sistema, en la que B se almacena como su columna $(N + 1) - ésima$:

$$AX = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \dots & a_{1N}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & a_{23}^{(1)} & \dots & a_{2N}^{(1)} \\ a_{31}^{(1)} & a_{32}^{(1)} & a_{33}^{(1)} & \cdots & a_{3N}^{(1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{N1}^{(1)} & a_{N2}^{(1)} & a_{N3}^{(1)} & \cdots & a_{NN}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} a_{1\,N+1}^{(1)} \\ a_{2\,N+1}^{(1)} \\ a_{3\,N+1}^{(1)} \\ \vdots \\ a_{N\,N+1}^{(1)} \end{bmatrix} = B$$

Paso 1. Almacenamos todos los coeficientes en la matriz ampliada. El superíndice (1) indica que ésta es la primera vez que se almacena un número en la posición (r, c).

$$\left[\begin{array}{cccc|c} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \cdots a_{1N}^{(1)} & a_{1\,N+1}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & a_{23}^{(1)} & \dots a_{2N}^{(1)} & a_{2\,N+1}^{(1)} \\ a_{31}^{(1)} & a_{32}^{(1)} & a_{33}^{(1)} & \dots a_{3N}^{(1)} & a_{3\,N+1}^{(1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{N1}^{(1)} & a_{N2}^{(1)} & a_{N3}^{(1)} & \dots a_{NN}^{(1)} & a_{N\,N+1}^{(1)} \end{array}\right]$$

Paso 2. Si es necesario, intercambiamos filas de manera que $a_{11}^{(1)} \neq 0$; entonces se elimina la incógnita x_1 en todas las filas desde la segunda hasta la última. En este proceso, m_{r1} es el número por el que hay que multiplicar la primera fila para restarla de la fila r-ésima. Los nuevos $a_{rc}^{(2)}$ se superindizan con un (2) para señalar que ésta es la segunda vez que se almacena un número en la posición (r, c) de la matriz.

El resultado tras el paso 2 es

$$\left[\begin{array}{cccc|c} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \cdots a_{1N}^{(1)} & a_{1\,N+1}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \ldots a_{2N}^{(2)} & a_{2\,N+1}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} & \ldots a_{3N}^{(2)} & a_{3\,N+1}^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & a_{N2}^{(2)} & a_{N3}^{(2)} & \ldots a_{NN}^{(2)} & a_{N\,N+1}^{(2)} \end{array}\right]$$

Paso3. Si es necesario, intercambiamos la segunda fila con alguna posterior para que $a_{22}^{(2)} \neq 0$; luego, eliminamos la incógnita $x_2$en todas las filas desde la tercera hasta la última. En este proceso, m_{r2} es el número por el que hay que multiplicar la segunda fila para restarla de la fila r-ésima. Los nuevos elementos $a_{rc}^{(3)}$ se superindizan con un (3) para señalar que ésta es la tercera vez que se almacena un número en la posición (r, c) de la matriz.

El resultado tras el paso 3 es

$$\left[\begin{array}{cccc|c} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \cdots a_{1N}^{(1)} & a_{1\,N+1}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \ldots a_{2N}^{(2)} & a_{2\,N+1}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & \ldots a_{3N}^{(3)} & a_{3\,N+1}^{(3)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & a_{N3}^{(3)} & \ldots a_{NN}^{(3)} & a_{N\,N+1}^{(3)} \end{array}\right]$$

Paso $q+1$. Este es el paso general. Si es necesario, intercambiamos la fila que ocupa el lugar q-ésimo con alguna posterior para que $a_{qq}^{(q)} \neq 0$; luego, eliminamos la incógnita $x_q$en todas las filas desde la $(q+1)$-ésima hasta la última. Ahora, m_{rq} es el número por el que hay que multiplicar la q-ésima para restarla de la fila r-ésima.

El resultado final es

$$\left[\begin{array}{cccc|c} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \cdots a_{1N}^{(1)} & a_{1\,N+1}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \ldots a_{2N}^{(2)} & a_{2\,N+1}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & \ldots a_{3N}^{(3)} & a_{3\,N+1}^{(3)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots a_{NN}^{(N)} & a_{N\,N+1}^{(N)} \end{array}\right]$$

Y el proceso de triangulación ya está terminado.

Puesto que A es invertible, cuando se realizan las operaciones con las filas, las matrices que se van obteniendo sucesivamente son también invertibles. Esto garantiza que $a_{kk}^{(k)} \neq 0$ para todo k a lo largo del proceso. Por tanto, podemos usar el algoritmo de sustitución regresiva para resolver $U\,X = Y$.

Pivoteo para evitar $\boldsymbol{a_{qq}^{(q)} = 0}$.

Cuando $a_{qq}^{(q)} = 0$, la fila q-ésima no puede usarse para eliminar los elementos de la columna q-ésima que están por debajo de la diagonal principal. Se hace necesario entonces hallar una fila posterior, digamos k-ésima con $k > q$, en la que $a_{kq}^{(q)} \neq 0$, e intercambiarlas para obtener un pivote no nulo. Este proceso se llama pivoteo y el criterio para decidir qué fila escoger se llama

estrategia de pivoteo. La estrategia de **pivoteo trivial** es la siguiente: Si $a_{qq}^{(q)} \neq 0$,entonces no se hace intercambio de fila; mientras que si $a_{qq}^{(q)} = 0$, entonces se localiza la primera fila por debajo de la q-ésima en la cual se tenga $a_{kq}^{(q)} \neq 0$ y se intercambia la fila q-ésima con la k-ésima. Esto proporciona el pivote no nulo deseado.

El propósito de las estrategias de pivoteo es usar como pivote el elemento de mayor magnitud y, una vez colocado en la diagonal principal, usarlo para eliminar los restantes elementos de su columna que están por debajo de él. Si en la columna q hay más de un elemento no nulo en la diagonal principal o por debajo de ésta, entonces hay varias formas de elegir qué filas se intercambian. La estrategia de **pivoteo parcial,** que consiste en lo siguiente: Para reducir la propagación de errores de redondeo, se sugiere que se compare el tamaño de todos los elementos de la columna q desde el que está en la diagonal hasta el de la última fila. Una vez localizada la fila, digamos k-ésima, en la que se encuentra el elemento de mayor error absoluto;o sea, si

$$|a_{kq}| = \max\{|a_{qq}|, |a_{q+1q}|, \dots, |a_{N-1q}|, |a_{NQ}|\} \qquad (22)$$

Entonces intercambiamos la fila q-ésima con la fila k-ésima, salvo que $k = q$; de esa manera, los multiplicadores m_{rq}, para $r = q+1, \dots, N$, serán todos menores que 1 en valor absoluto. Este proceso suele conservar las magnitudes relativas de los elementos de la matriz U del mismo orden que las de los coeficientes de la matriz original. Normalmente, la elección como pivote del mayor elemento también ayuda a que se propague un error más pequeño.

La técnica de **pivoteo parcial escalonado** puede usarse para reducir aún más los efectos de propagación de errores. En el pivoteo parcial escalonado se elige el elemento de la columna q-ésima, en o por debajo de la diagonal principal, que tiene un mayor tamaño relativo con respecto al resto de los elementos de su fila. Es decir, primero se busca en cada fila, desde la q-ésima hasta la última, el elemento de mayor tamaño, digamos s_r:

$$s_r = \max\{|a_{rq}|, |a_{rq+1}|, \dots, |a_{rN}|\} \quad para\ r = q, q+1, \dots, N. \qquad (23)$$

La fila pivote será la que ocupa el lugar k para el cual

$$\frac{|a_{kq}|}{s_k} = \max\left\{\frac{|a_{qq}|}{s_q}, \frac{|a_{q+1q}|}{s_{q+1}}, \dots, \frac{|a_{Nq}|}{s_N}\right\} \qquad (24)$$

Ahora se intercambian las filas q-ésima y la k-ésima, salvo que q=k. De nuevo, el propósito de esta estrategia de pivoteo es mantener las magnitudes relativas de los elementos de la matriz U del mismo orden que las de los coeicientes de la matriz original.

Factorización triangular.

Factorización LU.

Anteriormente hemos visto lo fácil que resulta resolver un sistema triangular superior. Ahora introduciremos el concepto de factorización triangular de una matriz: la posibilidad de escribir una matriz dada A como el producto de una matriz triangular inferior L, cuyos elementos en la

diagonal principal son todos iguales a 1, por una matriz triangular superior U, cuyos elementos diagonales son distintos de cero. Para facilitar la notación, ilustraremos los conceptos con matrices 4×4, pero se aplican a matrices de orden arbitrario $N \times N$.

Diremos que una matriz invertible A admite una factorización triangular o factorización LU si puede expresarse como el producto de una matriz triangular inferior L , cuyos elementos son todos iguales a 1, por una matriz triangular superior U:

$$A = LU, \qquad (25)$$

O, escrito de manera desarrollada,

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ l_{21} & 1 & 0 & 0 \\ l_{31} & l_{32} & 1 & 0 \\ l_{41} & l_{42} & l_{43} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{bmatrix}$$

La condición de que A sea invertible implica que $u_{kk} \neq 0$ para todo k.

En consecuencia, el algoritmo a seguir para el cálculo de los términos de las matrices $L\ y\ U$ será;

Para cada $k = 1,2, \dots, n$, hacer:

Para $j = k, k+1, \dots, n$ hacer;

$$u_{kj} = a_{kj} - \sum_{p=1}^{k-1} l_{kp} u_{pj} \qquad (26)$$

Para $i = k+1, k+2, \dots, n$ calcular:

$$l_{ik} = (a_{ik} - \sum_{p=1}^{k-1} l_{ip} u_{pk})/u_{kk} \qquad (27)$$

Solución de un sistema lineal.

Supongamos que la matriz de los coeficientes A de un sistema lineal $AX = B$ admite una factorización triangular como la de (25), entonces la solución de

$$LU\ X = B \qquad (28)$$

Puede obtenerse definiendo $Y = UX$ y resolviendo dos sistemas lineales:

$$primero\ se\ halla\ \ Y\ en\ LY = B \quad y\ luego\ X\ en\ UX = Y. \qquad (29)$$

En forma desarrollada, primero debemos resolver el sistema triangular inferior

$$\begin{array}{llll} y_1 & & & = b_1 \\ l_{21}y_1 + & y_2 & & = b_2 \\ l_{31}y_1 + & l_{32}y_2 + & y_3 & = b_3 \\ l_{41}y_1 + & l_{42}y_2 + & l_{43}y_3 + y_4 & = b_4 \end{array} \qquad (30)$$

Para obtener $y_1, y_2, y_3\ e\ y_4$ y, una vez que los tenemos, resolver el sistema triangular superior

$$\begin{aligned} u_{11}x_1 + u_{12}x_2 + u_{13}x_3 + u_{14}x_4 &= y_1 \\ u_{22}x_2 + u_{23}x_3 + u_{24}x_4 &= y_2 \\ u_{33}x_3 + u_{34}x_4 &= y_3 \\ u_{44}x_4 &= y_4 \end{aligned} \qquad (31)$$

Factorización de Cholesky.

Para matrices A que son equivalentes:

1. A es simétrica y definida positiva.
2. Existe L triangular inferior real, con elementos diagonales positivos, tal que $A = LL^t$.

Cálculo directo de la factorización.

En las hipótesis anteriores se define $A = LL^t$, por tanto será

$$a_{ij} = \sum_{k=1}^{n} l_{ik} l_{kj}^t = \sum_{k=1}^{h} l_{ik} l_{jk} \ \ siendo\ h = \min\{i, j\} \qquad (32)$$

Ahora para $i = j$ resulta

$$a_{jj} = \sum_{k=1}^{j} l_{jk}^2 = \sum_{k=1}^{j-1} l_{jk}^2 + l_{jj}^2$$

De donde se deduce que

$$l_{jj} = \sqrt{a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2} \qquad (33)$$

En tanto que para $i = j + 1, j + 2, \dots, n$ se tendrá

$$a_{ij} = \sum_{k=1}^{j} l_{ik} l_{jk} = \sum_{k=1}^{j-1} l_{ik} l_{jk} + l_{ij} l_{jj}$$

De la que se obtienen los

$$l_{ij} = (a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk}) / l_{jj} \qquad (34)$$

Una vez hallada la matriz L mediante el algoritmo descrito, se resolverían los dos sistemas triangulares

Una vez hallada la matriz L mediante el algoritmo descrito, se resolverían los dos sistemas triangulares $Ly = b$ y $L^t x = y$.

Métodos iterativos para sistemas lineales.

El objetivo de ésta sección es el extender a espacios de dimensión mayor que uno algunos de los métodos iterativos introducidos en el Capítulo2. Consideraremos extensiones del método de iteración de punto fijo que se aplican a sistemas de ecuaciones lineales.

En los métodos iterativos de resolución de sistemas lineales, se parte de una aproximación inicial de la solución $x^{(0)}$ del sistema $Ax = b$ y se generan las aproximaciones sucesivas $\{x^{(k)}\}_0^{\infty}$ en la forma

$$x^{(k+1)} = Ex^{(k)} + F \quad (k = 0,1,2,\dots) \qquad (35)$$

Donde $E \in M_n(R)$ se denomina matriz de iteración y $F \in M_{n\times 1}(R)$.

Un método como el descrito se dice convergente para el sistema $Ax = b$ *si* $\forall x^{(0)} \in R^n$: $\lim_{k\to\infty} x^{(k)} = x = A^{-1}b$.

Desde luego si el método es convergente la solución ha de verificar la siguiente relación

$$x = Ex + F$$

O lo que es equivalente

$$F = (I - E)A^{-1}b \qquad (36)$$

Aunque no verificase el recíproco, por ello se introduce la siguiente definición.

Un método (35) se dice consistente con el sistema $Ax = b$ si se verifica la condición (36).

Para toda norma matricial $||\cdot||$ y todo matriz $A \in M_n(R)$ se verifica que $\rho(A) \leq \big||A|\big|$. Además, se tiene que

$$\rho(A) = \inf\{\big||A|\big| : \big||\cdot|\big| \; es\; una\; norma\; matricial\}.$$

Con ayuda del cual puede probarse el teorema fundamental sobre la convergencia de métodos iterativos de resolución aproximada de sistemas lineales, que nos limitaremos a enunciar.

El método (35) es convergente con respecto al sistema $Ax = b$ si y solo si se verifican simultáneamente las dos condiciones siguientes:

1. El método es consistente con el sistema.
2. $\rho(E) < 1$.

Construcción de métodos iterativos.

Sea el sistema $Ax = b$ con A inversible cuya solución es $x = A^{-1}b$, descomponiendo A en la forma: $A = M - N$ con M inversible, entonces

$$Ax = b \leftrightarrow (M - N)x = Mx - Nx = b \leftrightarrow$$

$$\leftrightarrow Mx = Nx + b \leftrightarrow x = M^{-1}Nx + M^{-1}b$$

Expresión que sugiere el método

$$x^{(k+1)} = M^{-1}Nx^{(k)} + M^{-1}b \qquad (37)$$

Los métodos así construidos son consistentes por tanto, según el teorema fundamental, si y sólo si $\rho(M^{-1}N) < 1$.

Cota de error en los métodos iterativos.

Sea $x^{(k+1)} = Ex^{(k)} + F$ un método iterativo convergente para $Ax = b$, entonces $x^{(m-1)} - x = x^{(m-1)} - x^{(m)} + x^{(m)} - x$, lo cual implica

$$\left|\left|x^{(m-1)} - x\right|\right| \leq \left|\left|x^{(m-1)} - x^{(m)}\right|\right| + \left|\left|x^{(m)} - x\right|\right| \quad (38)$$

Como además

$$x^{(m)} - x = Ex^{(m-1)} + F - (Ex + F) = E(x^{(m-1)} - x)$$

Resulta que

$$\left|\left|x^{(m)} - x\right|\right| \leq ||E||\ \left|\left|x^{(m-1)} - x\right|\right| \quad (39)$$

De la que también se deduce que

$$\left|\left|x^{(m)} - x\right|\right| \leq |||E|||^{m} ||x^{(0)} - x||$$

Que puede servirnos para estimar a priori el número de iteraciones necesarias para una precisión deseada.

Ahora, combinando (38) y (39) es fácil obtener

$$(1 - ||E||)\ \left|\left|x^{(m-1)} - x\right|\right| \leq \left|\left|x^{(m)} - x^{(m-1)}\right|\right| \quad (40)$$

Y como el método es convergente, tomando una norma matricial tal que $||E|| < 1$ y una vectorial compatible con ella, de las desigualdades (39) y (40) se deduce

$$\left|\left|x^{(m)} - x\right|\right| \leq \frac{||E||}{1 - ||E||}\ \left|\left|x^{(m)} - x^{(m-1)}\right|\right| \leq \frac{||E||^{m}}{1 - ||E||}\ \left|\left|x^{(1)} - x^{(0)}\right|\right| \quad (41)$$

Que puede utilizarse para estimar el error y como test de parada del programa correspondiente.

Métodos iterativos particulares.

La expresión (37) anterior puede escribirse también en la forma

$$Mx^{(k+1)} = Nx^{(k)} + b \quad (42)$$

Donde $A = M - N$ y M inversible. Así pues, dado el sistema $Ax = b$, descompongamos la matriz $A = D - L - U$ con

$$D = \begin{pmatrix} a_{11} & 0 & 0 & & 0 \\ 0 & a_{22} & 0 & \cdots & 0 \\ 0 & 0 & a_{23} & & 0 \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn} \end{pmatrix}$$

$$L = \begin{pmatrix} 0 & 0 & 0 & & 0 \\ -a_{21} & 0 & 0 & \cdots & 0 \\ -a_{31} & -a_{32} & 0 & & 0 \\ & \vdots & & \ddots & \vdots \\ -a_{n1} & -a_{n2} & -a_{n3} & \cdots & 0 \end{pmatrix}$$

$$U = \begin{pmatrix} 0 & -a_{12} & -a_{13} & & -a_{1n} \\ 0 & 0 & -a_{23} & \cdots & -a_{2n} \\ 0 & 0 & 0 & & -a_{3n} \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix}$$

Método de Jacobi.

Supongamos que A es una matriz diagonal estrictamente dominante. Entonces el sistema de ecuaciones lineales $AX = B$ tienen solución única en $X = P$. Además, el proceso iterativo produce una sucesión de vectores $\{P_k\}$ que converge a P cualquiera que sea el vector de partida P_0.

*Se dice que una matriz A de orden $N \times N$ es diagonal estrictamente dominante cuando

$$|a_{kk}| > \sum_{\substack{j=1 \\ j \neq k}}^{N} |a_{kj}| \qquad para\ k = 1, 2, \dots, N. \quad (43)$$

Esto significa que en cada fila de la matriz, el tamaño del elemento que está en la diagonal principal debe ser mayor que la suma de los tamaños de todos los demás elementos de la fila.

El método consiste en hacer

$$M = D\ y\ N = L + U.$$

Como M debe ser inversible es necesario que todo $a_{ii} \neq 0$ para $i = 1,2, \dots, n$, con lo que puede expresarse de la forma

$$Dx^{(k+1)} = (L + U)x^{(k)} + b \quad (44)$$

Y en componentes resulta

$$x_j^{(k+1)} = \frac{b_j - a_{j1}x_1^{(k)} - \cdots - a_{jj-1}x_{j-1}^{(k)} - a_{jj+1}x_{j+1}^{(k)} - \cdots - a_{jN}x_N^{(k)}}{a_{jj}} \qquad (45)$$

Para $j = 1,2,.., N$.

En el método iterativo de Jacobi se usan todas las coordenadas del punto anterior en la obtención de coordenadas del punto nuevo, se dice que es un método de aproximaciones simultáneas. Este método será convergente si y sólo si $\rho(D^{-1}(L+U)) < 1$.

Método de Gauss-Seidel.

Algunas veces podemos acelerar la convergencia. Puesto que x_{k+1} es, probablemente, mejor aproximación al límite que x_k, sería razonable usar x_{k+1}en vez de x_k a la hora de calcular y_{k+1}y, de forma semejante, sería mejor usar $x_{k+1}\ e\ y_{k+1}$ en el cálculo de z_{k+1}. El método cosiste en tomar

$$M = D - L\ \ y\ N = U.$$

Con lo cual puede expresarse en forma vectorial como

$$(D-L)x^{(k+1)} = Ux^{(k)} + b \qquad (46)$$

Y en componentes se expresa en la forma

$$x_j^{(k+1)} = \frac{b_j - a_{j1}x_1^{(k)} - \cdots - a_{jj-1}x_{j-1}^{(k)} - a_{jj+1}x_{j+1}^{(k)} - \cdots - a_{jN}x_N^{(k)}}{a_{jj}} \qquad (47)$$

Para $j = 1,2,,,,N$.

En el método iterativo de Gauss-Seidel se emplean las coordenadas nuevas conforme se van generando, se dice que es un método de aproximaciones sucesivas.

Puede probarse que éste método también converge cuando la matriz A es diagonal estrictamente dominante así como para matrices simétricas definidas positivas. Este método será convergente si y sólo si $\rho((D-L)^{-1}U) < 1$.

Normalmente, el método de Gauss-Seidel converge más rápidamente que el de Jacobi, por lo que es el que se suele preferir. Se dan casos, sin embargo, en los que el método de Jacobi converge pero el de Gauss-Seidel no.

Ejemplo de aplicación ámbito naval.

Cálculo matricial de estructuras.

Una de las metodologías más potentes para el análisis de sistemas estructurales es la del método de las deformaciones. La relación entre las fuerzas aplicadas a una estructura y los desplazamientos que genera en la misma es a través de la llamada matriz de rigidez K de la estructura:

$$[F] = [K][\delta]$$

Donde F es el vector de fuerzas aplicadas en los nodos y δ es el vector de desplazamientos de los nudos. Los nodos son los extremos de las barras que componen la estructura. El proceso de resolución de una estructura mediante el cálculo matricial se inicia en la determinación de los términos matriz de rigidez de la estructura y fuerzas nodales. Posteriormente, se procede a la resolución numérica de la ecuación, resolviendo el sistema planteado mediante $[\delta] = [K]^{-1}[F]$.

En el caso de dos dimensiones, con tres grados de libertad por nodo, dos desplazamientos y un giro, la matriz de rigidez local de una barra de nodos empotrados o rígidos es:

$$[K]_L = \begin{bmatrix} \frac{EA}{L} & 0 & 0 & \frac{-EA}{L} & 0 & 0 \\ 0 & \frac{12EI}{L^3} & \frac{6EI}{L^2} & 0 & \frac{-12EI}{L^3} & \frac{6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{4EI}{L} & 0 & \frac{-6EI}{L^2} & \frac{2EI}{L} \\ \frac{-EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\ 0 & \frac{-12EI}{L^3} & \frac{-6EI}{L^2} & 0 & \frac{12EI}{L^3} & \frac{-6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{2EI}{L} & 0 & \frac{-6EI}{L^2} & \frac{4EI}{L} \end{bmatrix}$$

Las matrices de rigidez tienen las siguientes propiedades:

- Simétricas respecto a su diagonal principal.
- Todos los términos de la diagonal son valores mayores que cero.
- Son definidas positivas, es decir, su determinante es mayor que cero.
- Todos los términos dependen del material que constituye la barra, de propiedades de su sección transversal y de su longitud.

Ejercicio de aplicación de análisis matricial en dos dimensiones.

Sea la cuaderna de un barco con dos entrepuentes, apoyadas en las cubiertas y empotrada en el fondo. Otros datos son:

- B=Manga del buque= 20m.
- Bc=Manga de escotillas=10m.
- Carga sobre cubierta superior=2,5 T/m^2.

- Carga sobre la cubierta primer entrepuente=3,5 T/m^2.
- Carga sobre cubierta segundo entrepuente=3,5 T/m^2.
- E=2,1*10^9 N/m^2.
- Densidad agua del mar=1.026 t/m^3.
- Separación entre baos=0.6 m.

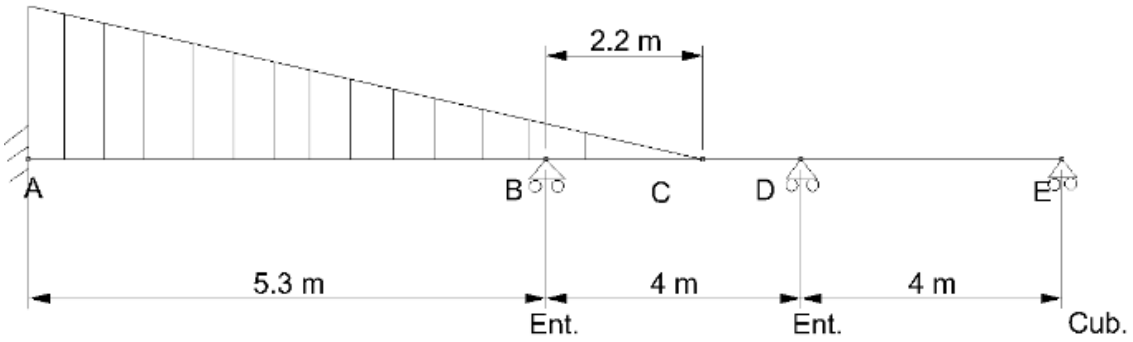


Figura 3.0

Tramo AB:

- Área sección perfil: 116.6 cm^2.
- Inercia sección: 6576 cm^4.

Tramo BCD:

- Área sección perfil: 100.5 cm^2.
- Inercia sección: 1701 cm^4.

Tramo CD:

- Área sección perfil: 60 cm^2.
- Inercia sección: 900 cm^4.

Solución:

Los baos apoyarán en las cuadernas y en las esloras que van bajo las brazolas de las escotillas, por lo que se considerará que reciben la mitad de la carga entre dos baos separados una distancia s=0.6 m.

$$0.5 * \frac{B - B_e}{2} * s * p\ Toneladas$$

Carga axial transmitida por la cubierta superior y entrepuentes a la cuaderna:

- En el tramo DE=3.75 Tons.
- En el tramo BD=3.75+5.25=9 Tons.
- En el tramo AB=3.75+5.25+5.75=14.25 Tons.

Momentos de empotramiento tramo AB:

$$P_A = \gamma h_A s = 1.026 * (5{,}3 + 2{,}2) * 0.6 = 4.617 \frac{T}{m} = 45293\ N/m$$

$$P_B = \gamma h_B s = 1.354 \frac{T}{m} = 13283\ N/m$$

$$m_A = -M_A = \frac{L^2}{60}(3p_A + 2p_B) = 76051\ N * m$$

$$m_B = M_B = -\frac{L^2}{60}(2p_A + 3p_B) = -61065\ N * m$$

Momentos de empotramiento tramo BC:

$$m_B = -M_B = -\frac{pL^2}{20} = 3214\ N * m$$

$$m_C = M_C = -\frac{pL^2}{30} = -2143\ N * m$$

Puntos 1y 2:

Una vez numerados los nodos se procede a la obtención de las matrices de rigidez de cada barra en ejes locales, se transforman a ejes globales y se ensamblan para obtener la matriz de rigidez global de la estructura. En este caso al estar todas las barras en dirección horizontal, las matrices locales y globales de cada barra coinciden, por lo que la matriz de cambio de coordenadas es una matriz unidad, es decir, $[K]_G = [T] * [K]_L * [T]^T = [K_L]$. Los grados de libertad de cada uno son u, v, θ comenzando por 1,2,3 para el primer nudo, 4,5,6 para el segundo y así hasta 13,14,15 para el quinto nudo.

Barra 1-2:

$$\frac{EA}{L} = \frac{2.1 * 10^9 * 116.6 * 10^{-4}}{5.3} = 4.62 * 10^8 N/m$$

$$\frac{12EI}{L^3} = \frac{12 * 2.1 * 10^9 * 6576 * 10^{-8}}{5.3^3} = 1.11310142 * 10^6\ N/m$$

$$\frac{6EI}{L^2} = 2.94971876 * 10^6\ N$$

$$\frac{4EI}{L} = 1.04223396 * 10^7\ Nm$$

$$\frac{2EI}{L} = 5.21116981 * 10^6\ Nm$$

Procedemos de igual manera para las barras restantes, con lo que habremos calculado los términos de nuestra matriz de rigidez en ejes globales $[K]_G$.

A continuación se calcula y se ensambla el vector de cargas. En primer lugar se aplican las cargas axiales en los nodos 2, 4 y 5, y posteriormente se calculan las cargas debidas al empotramiento de las vigas 1-2 y 2-3. La transformación de cargas en ejes locales a ejes globales se realiza mediante la matriz T, tal que $[f]_G = [T] * [f]_L$. En nuestro caso la matriz de transformación será unitaria.

Carga axial nodo 5:

$$N_{x5} = -3.75\ T = -3750 * 9.81 = -36787.5\ N = -3.67875\ E + 4\ N$$

Procedemos de igual manera y calculamos la carga axial en el resto de nodos.

Cargas en nodos de viga 1-2:

$$Momento_A = -m_A = \frac{L^2}{60}(3p_1 + 2p_2) = -76051\ N * m$$

$$Momento_B = -m_B = \frac{L^2}{60}(2p_1 + 3p_2) = 61065\ \ N * m$$

$$F_A = -R_A = -\left[\frac{L}{6}(2p_1 + p_2) - \frac{M_A - M_B}{L}\right] = -94578\ N$$

$$F_B = -R_B = -\left[\frac{L}{6}(p_1 + 2p_2) - \frac{M_A - M_B}{L}\right] = -60648\ \ N$$

Procedemos de igual manera y calculamos las cargas en los nodos del resto de vigas.

A continuación se imponen las condiciones de contorno en los nodos impedidos, lo que supone eliminar las filas y columnas de las matrices de rigidez y de cargas que corresponden con los grados de libertad impedidos, obteniéndose K_0 y f_0.

El sistema de ecuaciones $[K_0] * [d_0] = [f_0]$ que hay que resolver es el siguiente:

1421318182	0	-959318182	0	0	0	0	0	0
0	16947612	0	-4449050	3262636	0	0	0	0
-959318182	0	2131818182	0	0	-1172500000	0	0	0
0	-4449050	0	11429158	2197062	0	6646111	0	0
0	3262636	0	2197062	14500606	0	3987667	0	0
0	0	-1172500000	0	0	1487500000	0	-315000000	0
0	0	0	6646111	3987667	0	9865333	0	945000
0	0	0	0	0	-315000000	0	315000000	0
0	0	0	0	0	0	945000	0	1890000

$$* \begin{bmatrix} u_2 \\ \theta_2 \\ u_3 \\ v_3 \\ \theta_3 \\ u_4 \\ \theta_4 \\ u_5 \\ \theta_5 \end{bmatrix} = \begin{bmatrix} -139792 \\ 57851 \\ 0 \\ -4383 \\ 2143 \\ -88290 \\ 0 \\ -36787 \\ 0 \end{bmatrix}$$

Por lo que tenemos un sistema de ecuaciones lineales del tipo $AX = B$, que resolveremos aplicando los distintos métodos anteriormente descritos.

Para resolver el problema utilizaremos el método de Gauss con pivoteo total descrito anteriormente. Para ello iniciamos la interfaz gráfica y seleccionamos dicho método.

Para introducir los datos de nuestra matriz de coeficiente hacemos click en el botón “Generar matriz de coeficientes”, donde el programa nos remite a un archivo en el que debemos introducir los datos y donde se explica el modo de hacerlo.

Una vez que hemos introducido y guardado los datos de nuestra matriz de coeficientes hacemos click en el botón “Generar vector de términos independientes b” para introducir los términos independientes de nuestro sistema. El programa nos remite a otro archivo en el que podremos introducir dichos datos y en el que de igual modo se explica el modo de hacerlo. Seleccionaremos un máximo 5 cifras decimales a ver por pantalla.

Figura 3.1

Una vez que hemos introducido y guardado los datos necesarios hacemos click en el botón aplicar y aparece un archivo de texto que contiene los resultados obtenidos,

```
*************************************************************************
Matriz de coeficientes :
 1421318182.00000     0.00000     -959318182.00000     0.00000     0.00000     0.00000     0.00000     0.00000     0.00000
 0.00000     16947612.00000     0.00000     -4449050.00000     3262636.00000     0.00000     0.00000     0.00000     0.00000
-959318182.00000     0.00000     2131818182.00000     0.00000     0.00000     -1172500000.00000     0.00000     0.00000     0.00000
 0.00000     -4449050.00000     0.00000     11429158.00000     2197062.00000     0.00000     6646111.00000     0.00000     0.00000
 0.00000     3262636.00000     0.00000     2197062.00000     14500606.00000     0.00000     3987667.00000     0.00000     0.00000
 0.00000     0.00000     -1172500000.00000     0.00000     0.00000     1487500000.00000     0.00000     -315000000.00000     0.00000
 0.00000     0.00000     0.00000     6646111.00000     3987667.00000     0.00000     9865333.00000     0.00000     945000.00000
 0.00000     0.00000     0.00000     0.00000     0.00000     -315000000.00000     0.00000     315000000.00000     0.00000
 0.00000     0.00000     0.00000     0.00000     0.00000     0.00000     945000.00000     0.00000     1890000.00000
*************************************************************************
Matriz de términos independientes:
-139792.00000
 57851.00000
 0.00000
-4383.00000
 2143.00000
-88290.00000
 0.00000
-36787.00000
 0.00000
*************************************************************************
Vector solución x:              Incógnita número:

-0.00070                         3
-0.00057                         1
-0.00081                         6
-0.00093                         8
 0.00408                         2
-0.00078                         5
 0.00198                         4
-0.00107                         7
 0.00053                         9
```

Figura 3.2

Si ordenamos los resultamos obtenidos, podemos ver que nuestro vector solución es el siguiente;

$$\begin{bmatrix} u_2 \\ \theta_2 \\ u_3 \\ v_3 \\ \theta_3 \\ u_4 \\ \theta_4 \\ u_5 \\ \theta_5 \end{bmatrix} = \begin{bmatrix} -0.00057 \\ 0.00408 \\ -0.00070 \\ 0.00198 \\ -0.00078 \\ -0.00081 \\ -0.00107 \\ -0.00093 \\ 0.00053 \end{bmatrix}$$

Ayuda a la interfaz gráfica.

En éste último apartado desarrollaremos una explicación paso a paso de cómo utilizar la interfaz gráfica que ha sido implementada en el programa de análisis matemático Matlab para una mayor comprensión de la teoría descrita mediante ejemplos prácticos.

Para comenzar debemos iniciar el programa Matlab y situar la carpeta que contiene la interfaz gráfica, así como los demás programas a utilizar en éste capítulo, en la carpeta actual de Matlab denominada “Current folder” y hacer doble click sobre la carpeta “Interfaz gráfica” para que sea ésta a la que el programa recurra cuando realicemos una orden en la ventana de comandos.

Para comenzar introducimos la orden “guide” en la ventana de comandos y en la ventana emergente seleccionamos el fichero de nominado” Tema3_Sistemas de ecuaciones lineales”.

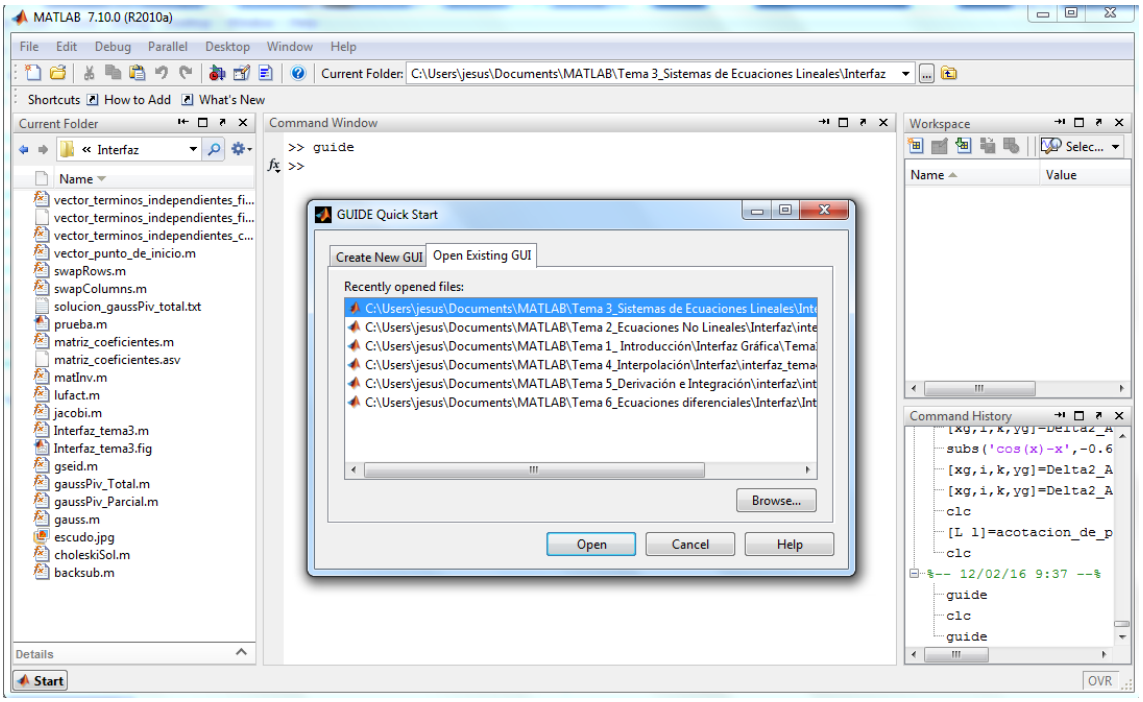


Figura 3.3

Al abrir dicho fichero aparecerá la interfaz de diseño de la propia interfaz gráfica, donde deberemos hacer doble click en el botón verde de reproducir o “play” situado en la zona superior.

Al hacerlo, aparecerá nuestra interfaz gráfica, inicialmente con la lista de métodos que han sido implementados y donde podremos seleccionar el método que queremos reproducir.

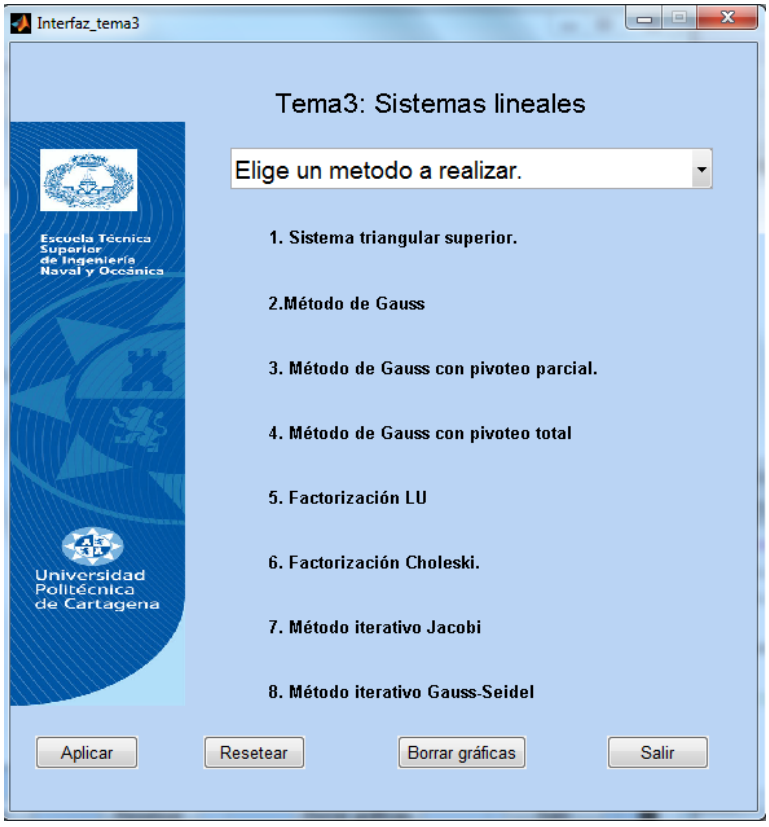


Figura 3.4

En este punto, pulsando en donde dice “Elige un método a realizar” nos aparece automáticamente una lista de los métodos de resolución de sistemas lineales, donde elegiremos el que deseemos llevar a cabo.

1. Sistema triangular superior.

Seleccionamos el método en la pantalla de inicio. Para comenzar, si presionamos el botón denominado “Sistema triangular superior” nos aparece en pantalla un pdf con la teoría y ejemplos necesarios para comprender el método.

Probaremos este método con el sistema lineal superior $Ax = b$, donde,

$$A = \begin{pmatrix} 3 & -2 & 1 & -1 \\ 0 & 4 & -1 & 2 \\ 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & 5 \end{pmatrix}, \quad b = \begin{pmatrix} 8 \\ -3 \\ 11 \\ 15 \end{pmatrix}$$

Que el programa resolverá utilizando el método de sustitución regresiva.

Para introducir los datos de nuestra matriz de coeficiente hacemos click en el botón “Generar matriz de coeficientes”, donde el programa nos remite a un archivo en el que debemos introducir los datos y donde se explica el modo de hacerlo.

Una vez que hemos introducido y guardado los datos de nuestra matriz de coeficientes hacemos click en el botón “Generar vector de términos independientes b” para introducir los términos independientes de nuestro sistema. El programa nos remite a otro archivo en el que podremos introducir dichos datos y en el que de igual modo se explica el modo de hacerlo. Seleccionaremos un máximo 3 cifras decimales a ver por pantalla.

Figura 3.5

Una vez que hemos introducido y guardado los datos del sistema de ecuaciones linales, al hacer click en el botón aplicar aparece un archivo de texto que contiene los resultados obtenidos,

```
1  ********************************************************************************
2  Matriz de coeficientes:
3   3.000    -2.000     1.000    -1.000
4   0.000     4.000    -1.000     2.000
5   0.000     0.000     2.000     3.000
6   0.000     0.000     0.000     5.000
7  ********************************************************************************
8  Vector de términos independientes:
9   8.000
10 -3.000
11  11.000
12  15.000
13 ********************************************************************************
14 Vector solución X:
15
16  2.000
17 -2.000
18  1.000
19  3.000
20
```

Figura 3.6

2. Método de Gauss.

Seleccionamos el método en la pantalla de inicio. Para comenzar, si presionamos el botón denominado “Método de Gauss” nos aparece en pantalla un pdf con la teoría y ejemplos necesarios para comprender el método.

Para probar este programa utilizaremos el sistema de ecuaciones lineales $Ax = b$ compuesto por;

$$A = \begin{pmatrix} 2 & 4 & 1 \\ 2 & 6 & -1 \\ 1 & 5 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 4 \\ 10 \\ 2 \end{pmatrix}$$

Este programa resolverá el sistema de ecuaciones lineales aplicando el método de Gauss sin aplicar ningún tipo de pivoteo.

Para introducir los datos de nuestro sistema, al igual que el caso anterior, pulsamos los botones “Generar matriz de coeficientes A” y “Generar vector términos independientes b” que nos remitirán a archivos en los que podremos introducir dichos datos.

Además queremos que aparezcan 4 cifras decimales por pantalla,

Figura 3.7

Una vez que hemos introducido y guardado los datos del sistema de ecuaciones lineales hacemos click en el botón aplicar y aparece un archivo de texto emergente con los resultados obtenidos,

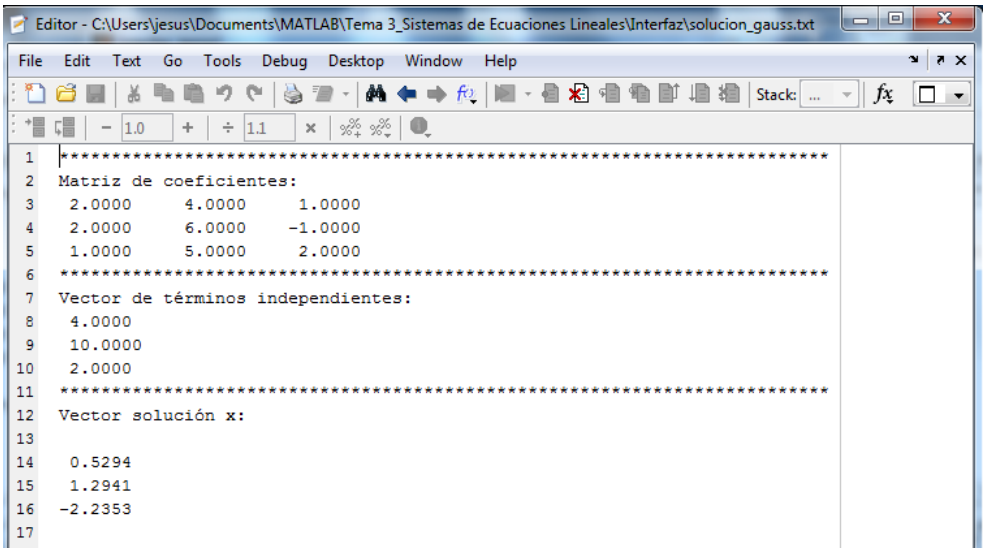


Figura 3.8

3. Método de Gauss con pivoteo parcial.

Seleccionamos el método en la pantalla de inicio. Para comenzar, si presionamos el botón denominado “Método de Gauss con pivote parcial” nos aparece en pantalla un pdf con la teoría y ejemplos necesarios para comprender el método.

Para probar este programa utilizaremos el sistema de ecuaciones lineales $Ax = b$ compuesto por;

$$A = \begin{pmatrix} 1 & 2 & 1 & 4 \\ 2 & 0 & 4 & 3 \\ 4 & 2 & 2 & 1 \\ -3 & 1 & 3 & 2 \end{pmatrix}, b = \begin{pmatrix} 13 \\ 28 \\ 20 \\ 6 \end{pmatrix}$$

Este programa resolverá el sistema de ecuaciones lineales aplicando el método de Gauss aplicando un pivoteo parcial en cada paso.

Para introducir los datos de nuestro sistema, al igual que el caso anterior, pulsamos los botones “Generar matriz de coeficientes A” y “Generar vector términos independientes b” que nos remitirán a archivos en los que podremos introducir dichos datos.

Además queremos que aparezca una cifra decimal por pantalla,

Figura 3.9

Una vez que hemos introducido y guardado los datos del sistema de ecuaciones lineales hacemos click en el botón aplicar y aparece un archivo de texto emergente con los resultados obtenidos,

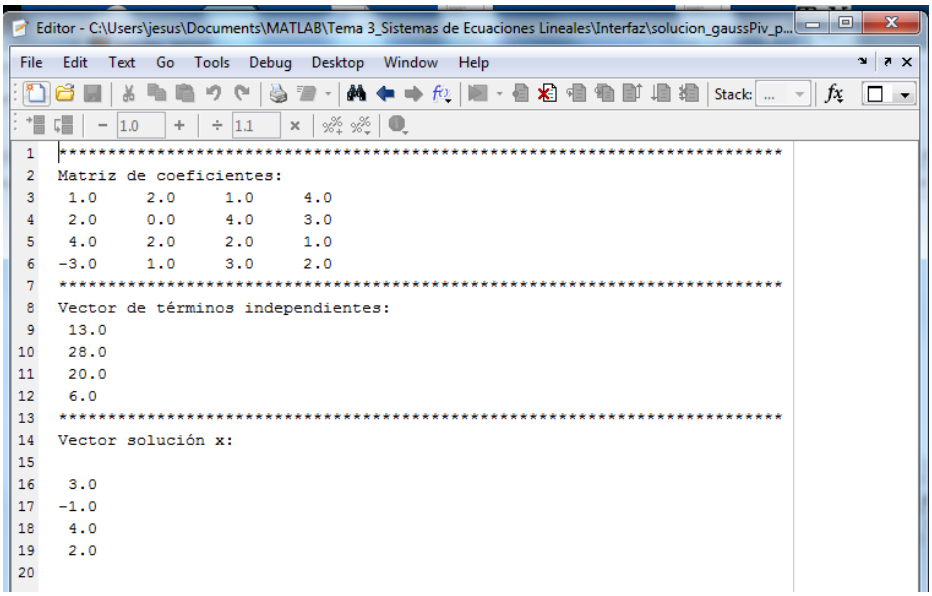


Figura 3.10

4. Método de Gauss con pivoteo total.

Seleccionamos el método en la pantalla de inicio. Para comenzar, si presionamos el botón denominado “Método de Gauss con pivote total” nos aparece en pantalla un pdf con la teoría y ejemplos necesarios para comprender el método.

Para probar este programa utilizaremos el sistema de ecuaciones lineales $Ax = b$ compuesto por;

$$A = \begin{pmatrix} 2 & 3 & -1 \\ 1 & -1 & 3 \\ 0 & 1 & -1 \end{pmatrix}, b = \begin{pmatrix} 4 \\ -4 \\ 2 \end{pmatrix}$$

Este programa resolverá el sistema de ecuaciones lineales aplicando el método de Gauss aplicando un pivoteo total en cada paso, es decir, buscando en cada paso el mejor pivote en el total de la submatriz correspondiente.

Para introducir los datos de nuestro sistema, al igual que el caso anterior, pulsamos los botones “Generar matriz de coeficientes A” y “Generar vector términos independientes b” que nos remitirán a archivos en los que podremos introducir dichos datos.

Además queremos que aparezca dos cifras decimales por pantalla,

Figura 3.11

Una vez que hemos introducido y guardado los datos del sistema de ecuaciones lineales hacemos click en el botón aplicar y aparece un archivo de texto emergente con los resultados obtenidos,

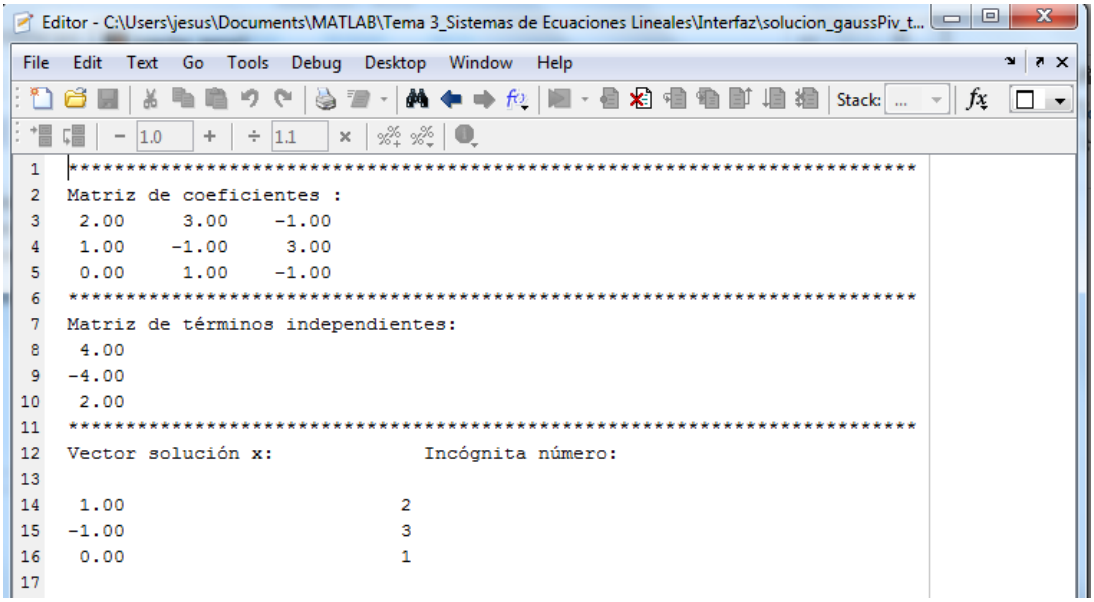


Figura 3.12

Debemos tener en cuenta que en la búsqueda del mejor pivote en cada paso, el orden de la filas se ha podido ver alterado, esto es indicado en la pantalla por el número de incógnita correspondiente a cada elemento del vector solución. Así, nuestro vector solución en este caso será;

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix}$$

5. Factorización LU.

Seleccionamos el método en la pantalla de inicio. Para comenzar, si presionamos el botón denominado "Factorización LU" nos aparece en pantalla un pdf con la teoría y ejemplos necesarios para comprender el método.

Para probar este programa utilizaremos el sistema de ecuaciones lineales $Ax = b$ compuesto por;

$$A = \begin{pmatrix} 1 & 4 & -2 \\ 3 & -2 & 5 \\ 2 & 3 & 1 \end{pmatrix}, b = \begin{pmatrix} 3 \\ 14 \\ 11 \end{pmatrix}$$

Para ello el programa aplica el proceso de factorización de la matriz A en LU, explicado en teoría y según el cual la matriz de coeficientes A puede descomponerse en $A = (L|U)$.

Para introducir los datos de nuestro sistema, al igual que el caso anterior, pulsamos los botones "Generar matriz de coeficientes A" y "Generar vector términos independientes b" que nos remitirán a archivos en los que podremos introducir dichos datos.

Además queremos que aparezca dos cifras decimales por pantalla,

Figura 3.13

Una vez que hemos introducido y guardado los datos del sistema de ecuaciones lineales hacemos click en el botón aplicar y aparece un archivo de texto emergente con los resultados obtenidos,

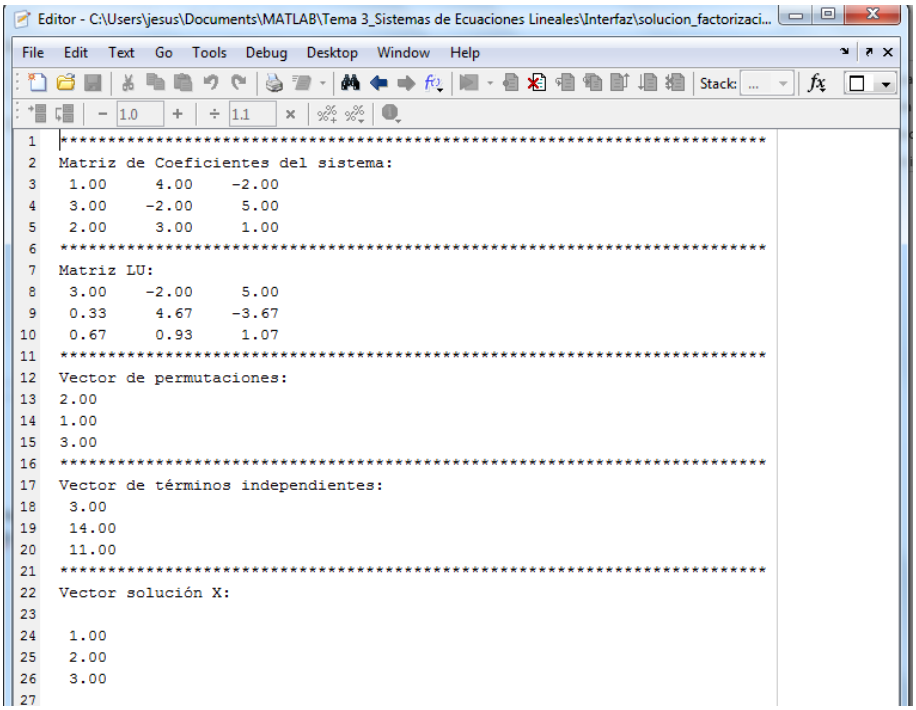


Figura 3.14

6. Factorización de Cholesky.

Seleccionamos el método en la pantalla de inicio. Para comenzar, si presionamos el botón denominado "Factorización Choleski" nos aparece en pantalla un pdf con la teoría y ejemplos necesarios para comprender el método.

Para probar este programa utilizaremos el sistema de ecuaciones lineales $Ax = b$ compuesto por;

$$A = \begin{pmatrix} 1 & -1 & 1 \\ -1 & 5 & 1 \\ 1 & 1 & 3 \end{pmatrix}, \qquad b = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

Recordemos que la matriz de coeficientes A debe ser definida positiva. Para resolver el sistema el programa aplica el proceso de factorización de la matriz A según Choleski, explicado en teoría y según el cual la matriz de coeficientes A puede descomponerse en $A = (L|L^t)$.

Para introducir los datos de nuestro sistema, al igual que el caso anterior, pulsamos los botones "Generar matriz de coeficientes A" y "Generar vector términos independientes b" que nos remitirán a archivos en los que podremos introducir dichos datos.

Además queremos que aparezca dos cifras decimales por pantalla,

Figura 3.15

Una vez que hemos introducido y guardado los datos del sistema de ecuaciones lineales hacemos click en el botón aplicar y aparece un archivo de texto emergente con los resultados obtenidos,

```
**********************************************************************
Matriz de coeficientes:
 1.00    -1.00     1.00
-1.00     5.00     1.00
 1.00     1.00     3.00
**********************************************************************
Vector de términos independientes:
 1.00
 1.00
 0.00
**********************************************************************
Matriz factorizada por Choleski L:
 1.00     0.00     0.00
-1.00     2.00     0.00
 1.00     1.00     1.00
**********************************************************************
Vector solución X:

 4.50
 1.50
-2.00

```

Figura 3.16

7. Método iterativo de Jacobi.

Seleccionamos el método en la pantalla de inicio. Para comenzar, si presionamos el botón denominado “Método iterativo de Jacobi” nos aparece en pantalla un pdf con la teoría y ejemplos necesarios para comprender el método.

Para probar este programa utilizaremos el sistema de ecuaciones lineales $Ax = b$ compuesto por;

$$A = \begin{pmatrix} 4 & -1 \\ 1 & 5 \end{pmatrix}, \qquad b = \begin{pmatrix} 15 \\ 9 \end{pmatrix}$$

A diferencia de los métodos anteriores, el método iterativo de Jacobi crea una sucesión de puntos $\{P_k\}$ de aproximaciones sucesivas a la solución. Por ello, debemos introducir además un punto en que comenzar las iteraciones P_0.

$$P_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Para introducir los datos de nuestro sistema, al igual que el caso anterior, pulsamos los botones “Generar matriz de coeficientes A” y “Generar vector términos independientes b” que nos remitirán a archivos en los que podremos introducir dichos datos. En esta ocasión debemos introducir el punto de inicio de las iteraciones. Para ello pulsamos el botón “Generar punto P” y el programa nos remite a un archivo en que podremos introducir los datos del punto de inicio.

Además queremos que realice un máximo de 20 iteraciones y que aparezcan dos cifras decimales por pantalla,

Figura 3.17

Una vez que hemos introducido y guardado los datos del sistema de ecuaciones lineales hacemos click en el botón aplicar y aparece un archivo de texto emergente con los resultados obtenidos,

```
1  ********************************************************************************
2  Matriz de coeficientes:
3   4.00    -1.00
4   1.00     5.00
5  ********************************************************************************
6  Vector de términos independientes:
7   15.00
8   9.00
9  ********************************************************************************
10 Punto de inicio P:
11  0.00
12  0.00
13 ********************************************************************************
14 Número de iteraciones máximas a realizar:
15 20
16 ********************************************************************************
17 Tolerancia permitida:
18 2.220446e-010
19 ********************************************************************************
20 Número de iteraciones realizadas:
21 16
22 ********************************************************************************
23 Vector solución X:
24
25  4.00
26  1.00
27
```

Figura 3.18

8. Método iterativo de Gauss-Seidel.

Seleccionamos el método en la pantalla de inicio. Para comenzar, si presionamos el botón denominado “Método iterativo de Gauss-Seidel” nos aparece en pantalla un pdf con la teoría y ejemplos necesarios para comprender el método.

Para probar este programa utilizaremos el mismo sistema de ecuaciones lineales $Ax = b$, utilizado en el método anterior para así poder comparar diferencias entre los resultados obtenidos.

Al igual que en caso anterior, el método iterativo de Gauss-Seidel crea una sucesión de puntos $\{P_k\}$ de aproximaciones sucesivas a la solución. Por ello, debemos introducir además un punto en que comenzar las iteraciones P_0.

$$P_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Para introducir los datos de nuestro sistema, al igual que el caso anterior, pulsamos los botones “Generar matriz de coeficientes A” y “Generar vector términos independientes b” que nos remitirán a archivos en los que podremos introducir dichos datos. Al igual que en caso anterior, debemos introducir el punto de inicio de las iteraciones. Para ello pulsamos el botón “Generar punto P” y el programa nos remite a un archivo en que podremos introducir los datos del punto de inicio.

Además queremos que realice un máximo de 20 iteraciones y que aparezcan dos cifras decimales por pantalla,

Figura 3.19

Una vez que hemos introducido y guardado los datos del sistema de ecuaciones lineales hacemos click en el botón aplicar y aparece un archivo de texto emergente con los resultados obtenidos,

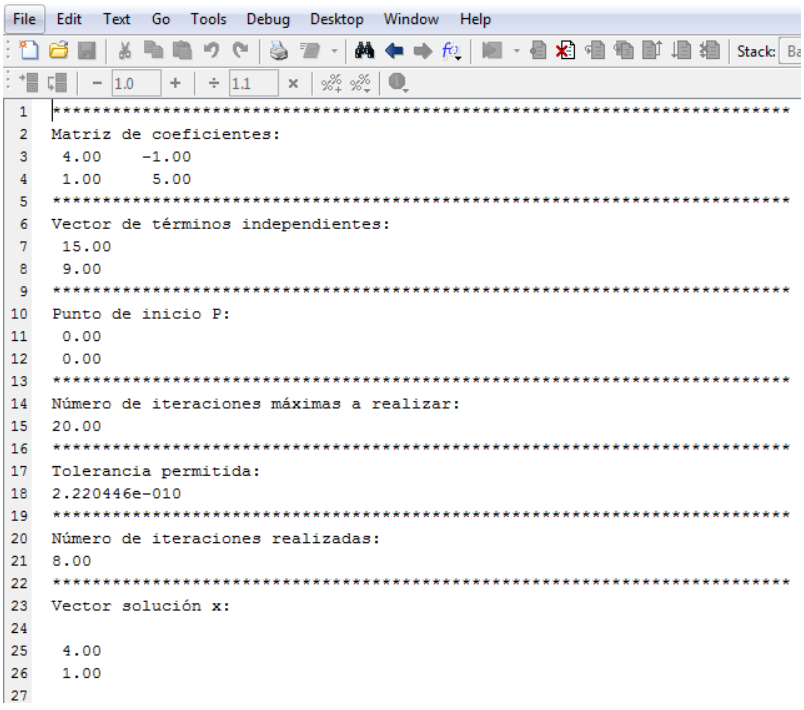


Figura 3.20

Como podemos observar, llegamos a la misma solución del caso anterior, sin embargo en este método se ha requerido la mitad de iteraciones que en caso anterior. Por ello podemos afirmar, tal y como enunciamos en teoría, que el método de Gauss-Seidel converge más rápidamente que el método de Jacobi salvo excepciones.

Capítulo 4. Interpolación polinomial.

Introducción.

En el tema de introducción vimos cómo puede usarse un polinomio de Taylor para aproximar la función $f(x)$. La información necesaria para construir el polinomio de Taylor es el valor de f y los de sus derivadas en x_0. Un inconveniente de este procedimiento es que debemos conocer las derivadas de orden superior y, a menudo, suele ocurrir que bien no están disponibles, o bien son difíciles de calcular.

Supongamos que conocemos $N+1$ puntos $(x_0, y_0), (x_1, y_1), \dots, (x_N, y_N)$ de la curva $y = f(x)$, donde las abscisas x_k se distribuyen en un intervalo $[a, b]$ de manera que

$$a \le x_0 < x_1 < \cdots < x_N \le b \quad e \;\; y_k = f(x_k).$$

Construiremos un polinomio $P(x)$ de grado N que pase por los $N+1$ puntos. Para construirlo, únicamente necesitaremos conocer los valores $x_k \; e \;\; y_k$, así que las derivadas de orden superior no nos harán falta. El polinomio $P(x)$ puede luego usarse como aproximación a $f(x)$ en todo el intervalo $[a, b]$; no obstante, si queremos conocer la función error $E(x) = f(x) - P(x)$, entonces sí necesitaremos conocer $f^{(N+1)}(x)$ o bien la cota de error de su tamaño como

$$M = \max\{|f^{(N+1)}(x)| : a \le x \le b\}.$$

Existen funciones especiales $y = f(x)$, que aparecen en análisis de tipo estadístico o científico, para las que sólo disponemos de una tabla de valores; es decir, sólo conocemos $N+1$ puntos (x_k, y_k) y es necesario dar un método para aproximar $f(x)$ en abscisas que no están tabuladas. Si el error de los valores tabulados es significativo, entonces es mejor usar los métodos de ajuste de curvas. Si por el contrario, los puntos (x_k, y_k) tienen un alto grado de precisión, entonces podemos considerar el polinomio $y = P(x)$ que pasa por todos ellos. Cuando $x_0 < x < x_N$, la aproximación $P(x)$ se conoce como valor interpolado; si se tiene $x < x_0$ o bien $x_N < x$, entonces $P(x)$ se conoce como valor extrapolado.

Los tipos más usuales de problemas de interpolación son los que se describen a continuación.

1. Interpolación polinomial de Lagrange. Consiste en dados los valores de una función f, $f_k = f(x_k)$ $(k = 0,1,2, \dots, n)$, en $n+1$ puntos distintos $\{x_k\}_0^N$ del intervalo $[a, b]$, determinar, si existe, un polinomio $P(x)$ de grado menor o igual a n tal que $P(x_k) = f_k$ $(k = 0,1,2, \dots, n)$.
2. Fórmula de Newton. Por el método de diferencias divididas
3. Interpolación de Taylor. En este problema se suponen conocidos los valores de la función f y sus derivadas sucesivas hasta el orden n en el punto x_0 y se trata de hallar un polinomio $P(x)$ de grado menor o igual que n tal que $p^{(k)}(x_0) = f^{(k)}(x_0)$ $(k = 0,1,2, \dots, n)$.
4. Interpolación de Hermite. Ahora se suponen conocidos los valores de la función f y su derivada f' en los puntos $x_0, x_1, .., x_n$,que abreviaremos por $f_k y\, f'_k$ $(para\ i = 0,1,2, .. n)$ y se trata de hallar un polinomio $P(x)$, de grado menor o igual a $2n+1$ tal que $P(x_K) = f_k$ y $P'(x_k) = f'_k$ $(para\ k = 0,1,2, \dots, n)$
5. Funciones cerchas interpoladoras. Splines cúbicos.

Interpolación de Lagrange.

Interpolar significa estimar el valor desconocido de una función en un punto, tomando una medida ponderada de sus valores conocidos en puntos cercanos al dado. En la interpolación lineal, también conocida como regla de tres, se utiliza un segmento rectilíneo que pasa por dos puntos (x_0, y_0) y (x_1, y_1) viene dada por $m = \frac{y_1 - y_0}{x_1 - x_0}$ así en la ecuación de la recta escrita como $y = m(x - x_0) + y_0$ podemos sustituir m y obtener

$$y = P(x) = y_0 + (y_1 - y_0)\frac{x - x_0}{x_1 - x_0} \qquad (1)$$

Si desarrollamos esta fórmula, el resultado es un polinomio de grado menor o igual que uno y la evaluación de $P(x)$ en x_0 y $x_1$produce $y_0\ e\ y_1$, respectivamente:

$$\begin{aligned} P(x_0) &= y_0 + (y_1 - y_0)(0) = y_0, \\ P(x_1) &= y_0 + (y_1 - y_0)(1) = y_1. \end{aligned} \qquad (2)$$

El matemático francés Joseph Louis Lagrange descubrió que se puede encontrar este polinomio usando un método ligeramente distinto. Si escribimos

$$y = P_1(x) = y_0\frac{x - x_1}{x_0 - x_1} + y_1\frac{x - x_0}{x_1 - x_0}, \qquad (3)$$

Entonces cada uno de los sumandos del miembro derecho de esta relación es un término lineal, por lo que su suma será un polinomio de grado menor o igual que uno. Denotemos los coeficientes de (3) por

$$L_{1,0}(x) = \frac{x - x_1}{x_0 - x_1} \quad y \quad L_{1,1}(x) = \frac{x - x_0}{x_1 - x_0}. \qquad (4)$$

Un sencillo cálculo muestra que $L_{1,0}(x_0) = 1$,$L_{1,0}(x_1) = 0, L_{1,1}(x_0) = 0$ y $L_{1,1}(x_1) = 1$, así que el polinomio $P_1(x)$ definido en (3) también pasa por los dos puntos dados:

$$P_1(x_0) = y_0 + y_1(0) = y_0 \quad y \quad P_1(x_1) = y_0(0) + y_1 = y_1. \qquad (5)$$

Los términos $L_{1,0}(x)\ y\ L_{1,1}(x)$ definidos en (4) se llaman polinomios coeficientes de Lagrange para los nodos $x_0\ y\ x_1$. Usando esta notación podemos escribir (3) como una suma

$$P_1(x) = \sum_{k=0}^{1} y_k L_{1,k}(x). \qquad (6)$$

La forma de generalizar esta expresión para construir un polinomio $P_N(x)$ que tenga grado menor o igual que N y que pase por $N + 1$ puntos $(x_0, y_0), (x_1, y_1), \dots, (x_N, y_N)$ es la fórmula

$$P_N(x) = \sum_{k=0}^{N} y_k L_{N,k}(x), \qquad (7)$$

Donde $L_{N,k}$ es el polinomio coeficiente de Lagrange para los nodos $x_N, x_1, \dots, x_N$ definido por

$$L_{N,k}(x) = \frac{(x-x_0)\cdots(x-x_{k-1})(x-x_{k+1})\cdots(x-x_N)}{(x_k-x_0)\cdots(x_k-x_{k-1})(x_k-x_{k+1})\cdots(x_k-x_N)}, \qquad (8)$$

Donde se sobreentiende que los factores $(x-x_k)y\ (x_k-x_k)$ no aparecen en el coeficiente del miembro derecho de la relación. Resulta cómodo introducir la notación compacta para el producto y escribir

$$L_{N,k}(x) = \frac{\prod_{\substack{j=0 \\ j\neq k}}^{N}(x-x_j)}{\prod_{\substack{j=0 \\ j\neq k}}^{N}(x_k-x_j)}. \qquad (9)$$

Esta notación indica que en el numerador se forma el producto de todos los actores lineales $(x-x_j)$ pero sin incluir el factor $(x-x_k)$.

Por lo tanto, el polinomio de interpolación buscado,$P_N(x_j)$, se obtiene como combinación lineal de los polinomios de base de Lagrange en la forma

$$P_N(x_j) = y_0 L_{N,0}(x_j) + \cdots + y_j L_{N,j}(x_j) + \cdots + y_N L_{N,N}(x_j) \qquad (10)$$

Para probar que $P_N(x)$ es único, aplicamos el teorema fundamental del álgebra, el cual establece que un polinomio no nulo $T(x)$ de grado menor o igual que N tiene, como mucho, N raíces. Supongamos entonces que hay otro polinomio $Q_N(x)$ de grado menor o igual que N cuya gráfica pasa también por los puntos $N+1$ puntos dados. Formando el polinomio diferencia $T(x) = P_N(x) - Q_N(x)$, vemos que $T(x)$ es de grado menor o igual que N y que $T(x_j) = P_N(x_j) - Q_N(x_j) = y_j - y_j = 0$ para cada $j = 0,1,\ldots,N$; por tanto $T(x) \equiv 0$ y, en consecuencia, $Q_N(x) = P_N(x)$.

Otra forma, más conocida, para demostrar que $P_N(x)$ es único es, usando la base $\{1, x, \ldots, x^n\}$ el determinante de la matriz de coeficientes es el denominado determinante de Vandermonde.

$$det\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & & x_1^n \\ & \vdots & & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix} = \prod_{i>j}(x_i - x_j)$$

det≠0 sí y sólo si las abscisas x_i no se repiten, por lo que no hay dos iguales.

Términos y cotas de error.

Es importante el entender la naturaleza del término del error que se comete cuando se utiliza un polinomio interpolador de Lagrange para aproximar una función $f(x)$.

Supongamos que $f \in C^{N+1}[a,b]$ y que $x_0, x_1, \ldots, x_N \in [a,b]$ son $N+1$ nodos de interpolación.

Si $x \in [a,b]$, entonces

$$f(x) = P_N(x) + E_N(x) \qquad (11)$$

Donde el término del error se puede escribir como

$$E_N(x) = \frac{(x-x_0)(x-x_1)\cdots(x-x_N)f^{(N+1)}(c)}{(N+1)!}, \qquad (12)$$

Para algún valor $c = c(x)$ del intervalo $[a, b]$.

Polinomio interpolador de Newton.

Hay ocasiones en las que resulta útil construir varios polinomios aproximantes $P_1(x), P_2(x), \ldots, P_N(x)$y, después, elegir el más adecuado a nuestras necesidades. Si usamos los polinomios interpoladores de Lagrange, uno de los inconvenientes es que no hay relación entre la construcción de $P_{N-1}(x)$ y la de $P_N(x)$, cada polinomio debe construirse individualmente y el trabajo necesario para calcular polinomios de grado elevado requiere hacer muchas operaciones. Vamos a seguir ahora un camino de construcción distinto, en el cual los polinomios interpoladores, que se llamarán de Newton, se calculan mediante un esquema recursivo

$$P_1(x) = a_0 + a_1(x-x_0) \qquad (13)$$

$$P_2(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) \qquad (14)$$

$$P_3(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) + a_3(x-x_0)(x-x_1)(x-x_2) \qquad (15)$$

…

$$P_N(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) + a_3(x-x_0)(x-x_1)(x-x_2) + a_N(x-x_0)(x-x_1)(x-x_2)\ldots(x-x_{N-1}) \qquad (16)$$

El polinomio $P_N(x)$ se obtiene a partir de $P_{N-1}(x)$ usando la recurrencia

$$P_N(x) = P_{N-1}(x) + a_N(x-x_0)(x-x_1)(x-x_2)\cdots(x-x_{N-1}) \qquad (17)$$

En este marco se dice que el polinomio $P_N(x)$ dado en la fórmula (16) es un polinomio de Newton con N centros $x_0, x_1, \ldots, x_{N-1}$. Puesto que $P_N(x)$ involucra sumas de productos factoriales lineales, siendo

$$a_N(x-x_0)(x-x_1)(x-x_2)\cdots(x-x_{N-1})$$

El de grado mayor, está claro que $P_N(x)$ es un polinomio de grado menor o igual que N.

Supongamos que queremos encontrar los coeficientes a_k de todos los polinomios $P_1(x), \ldots, P_N(x)$ que nos sirven para aproximar una función dada $f(x)$. Entonces cada $P_k(x)$ es el polinomio de Newton que tiene como centros los puntos $x_0, x_1, \ldots, x_k$ y es también el polinomio de interpolación para los nodos $x_0, x_1, \ldots, x_{k+1}$. Para el polinomio $P_1(x)$, los coeficientes a_0 y a_1 tienen un significado familiar; en este caso, se tiene que

$$P_1(x_0) = f(x_0) \quad \text{y} \quad P_1(x_1) = f(x_1). \qquad (18)$$

De modo que, usando (13) y (18), podemos despejar a_0 y obtener

$$f(x_0) = P_1(x_0) = a_0 + a_1(x_0 - x_0) \qquad (19)$$

Por tanto, $a_1$es la pendiente de la línea recta que pasa por los puntos $(x_0, f(x_0))$ y $\big(x_1, f(x_1)\big)$.

Los coeficientes $a_0\, y\, a_1$ son los mismos para $P_1(x) y\, P_2(x)$ así que, para continuar, ahora evaluamos la expresión (2) en el nodo x_2 y obtenemos

$$f(x_2) = P_2(x_2) = a_0 + a_1(x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1) \qquad (20)$$

Usando en (20) los valores de $a_0\ \ y\, a_1$ calculados anteriormente, nos queda

$$a_2 = \frac{\left(\frac{f(x_2) - f(x_1)}{x_2 - x_1} \_ \frac{f(x_1) - f(x_0)}{x_1 - x_0}\right)}{(x_2 - x_0)} \qquad (21)$$

Donde el numerador es la diferencia entre un coeficiente de diferencias, vamos a precisar esta idea de diferencias divididas, que será la herramienta con la cual podremos continuar el proceso recursivo.

Las diferencias divididas de una función $f(x)$ se definen como:

$$\begin{gathered} f[x_k] = f(x_k) \\ f[x_{k-1}, x_k] = \frac{f[x_k] - f[x_{k-1}]}{x_k - x_{k-1}} \\ f[x_{k-2}, x_{k-1}, x_k] = \frac{f[x_{k-1}, x_k] - f[x_{k-2}, x_{k-1}]}{x_k - x_{k-2}} \end{gathered} \qquad (22)$$

Las diferencias divididas de orden superior se forman de acuerdo con la siguiente regla recursiva:

$$f\left[x_{k-j}, x_{k-j+1}, \dots, x_k\right] = \frac{f\left[x_{k-j+1}, \dots, x_k\right] - f[x_{k-j}, \dots, x_{k-1}]}{x_k - x_{k-j}}, \qquad (23)$$

Regla que se utiliza para construir la tabla de diferencias divididas.

Los coeficientes a_k de los polinomios $P_N(x)$ dependen de los valores de interpolación $f(x_j)$ $(con\ j = 0,1, \dots, k)$; el siguiente teorema establece que a_k puede calcularse usando las diferencias divididas de $f(x)$:

$$a_k = f[x_0, x_1, \dots, x_k] \qquad (23)$$

Supongamos que $x_0, x_1, \dots, x_N$ son $N + 1$ números distintos en $[a, b]$. Entonces existe un único polinomio $P_N(x)$ de grado menor o igual que N tal que

$$f(x_j) = P_N(x_j) \qquad para\ j = 0,1, \dots, N.$$

La forma de Newton de este polinomio interpolador es

$$P_N(x) = a_0 + a_1(x - x_0) + \cdots + a_N(x - x_0)(x - x_1) \cdots (x - x_{N-1}) \qquad (24)$$

Siendo $a_k = f[x_0, x_1, \dots, x_k]$ para $k = 0,1, \dots, N$.

Si usamos el polinomio descrito en (24) para aproximar una función f, si $f \in C^{N+1}[a, b]$, entonces para cada $x \in [a, b]$ existe un número $c = c(x)$ en (a, b), tal que el término del error puede escribirse como

$$E_N(x) = \frac{(x - x_0)(x - x_1) \cdots (x - x_N) f^{(N+1)}(c)}{(N + 1)!} \qquad (25)$$

El término de error $E_N(x)$es, naturalmente, el mismo que el polinomio interpolador de Lagrange que vimos en la fórmula (12).

Interpolación de Taylor.

En este caso, a diferencia de lo explicado en el capítulo de introducción, en el que deseábamos encontrar un polinomio $P_N(x)$ que aproximara a una cierta función $f(x)$ cerca de un cierto punto x_0, en este caso, tenemos un punto $x_0 \in R$ y suponemos conocidos los valores de la función f y sus derivadas sucesivas hasta el orden n en el punto x_0 y se trata de hallar un polinomio $P_N(x)$ de grado menor o igual que n tal que $p^{(k)}(x_0) = f^{(k)}(x_0)$ $(k = 0,1,2,..,n)$

Recordemos la expresión del polinomio de Taylor para aproximar una cierta función $f \in C^{N+1}[a,b]$ cerca de un punto $x_0 \in [a,b]$. Si $x \in [a,b]$, entonces

$$f(x) = P_N(x) + E_N(x), \qquad (26)$$

Donde $P_N(x)$ es un polinomio que podemos utilizar para aproximar $f(x)$:

$$f(x) \approx P_N(x) = \sum_{k=0}^{N} \frac{f^{(k)}(x_0)}{k!} (x - x_0)^k, \qquad (27)$$

Llamado polinomio de Taylor de grado N de f alrededor de x_0, y el término del error

$$E_N(x) = \frac{f^{(N+1)}(c)}{(N+1)!}(x - x_0)^{N+1} \qquad (28)$$

Para algún valor de $c = c(x)$ que está entre $x\ y\ x_0$.

Pues bien, entonces si $P_N(x)$ es el polinomio de Taylor de grado N, debe cumplir que

$$P_N^{(k)}(x_0) = f^{(k)}(x_0) \quad para\ k = 0,1,\dots,N. \qquad (29)$$

Por ejemplo si ponemos $x = x_0$ en las ecuaciones (27) y (28) el resultado es $P_N(x_0) = f(x_0)$. Esto prueba que (16) es cierto para k=0. Derivando el miembro derecho de (27) obtenemos

$$P'_N(x) = \sum_{k=1}^{N} \frac{f^{(k)}(x_0)}{(k-1)!}(x - x_0)^{k-1} = \sum_{k=0}^{N-1} \frac{f^{(k+1)}(x_0)}{k!}(x - x_0)^k. \qquad (30)$$

Ponemos $x = x_0$ en (17) y obtenemos $P'_N(x_0) = f'(x_0)$. Esto prueba que (29) es cierto para k=1. Derivando sucesivamente en (30) se obtienen las demás igualdades de (29).

Es decir que si tenemos una serie de valores reales $w_0, w_1, w_2, \dots w_{(N-1)}$ que son los valores de las sucesivas derivadas toman en un punto x_0 entonces existe un único polinomio $P(x)$ tal que

$$P^{(k)}(x_0) = w_k \qquad \forall k = 0,1,2\dots,N \qquad (31)$$

La base dual que denotaremos por $\{t_0, \dots, t_N\}$, viene dada por:

$$t_k(x) = \frac{(x - x_0)^{(k)}}{k!}$$

Entonces

$$P_N(x) = \sum_{k=0}^{N} w_k t_k(x) \qquad (31)$$

Interpolación de Hermite.

La interpolación de Hermite consiste en encontrar un polinomio de grado requerido que satisfaga sobre cada punto x_i las condiciones $p^{(k)}(x_i) = f^{(k)}(x_i), \forall\, k = 1, \dots, n_i$, donde el número de derivadas consecutivas consideradas puede depender de x_i, es decir, se puede considerar un número diferentes de derivadas en cada punto.

De manera resumida, podemos plantear el resultado de la siguiente manera. Encontrar un polinomio $p(x)$ que satisfaga,

$$\begin{array}{ccccc} x_0 & & x_1 & & x_m \\ p(x_0) = f(x_0) & & p(x_1) = f(x_1) & & p(x_m) = f(x_m) \\ p'^{(x_0)} = f'(x_0) & & p'^{(x_1)} = f'(x_1) & & p'(x_m) = f'(x_m) \\ p''^{(x_0)} = f''(x_0) & \cdots & p''^{(x_1)} = f''(x_1) & \cdots & p''(x_m) = f''(x_m) \\ \vdots & & \vdots & & \vdots \\ p^{(n_0)}(x_0) = p^{(n_0)}(x_0) & & p^{(n_0)}(x_1) = p^{(n_0)}(x_1) & & p^{(n_m)}(x_m) = p^{(n_m)}(x_m) \end{array} \qquad (32)$$

Reiteramos que las derivadas deben ser consecutivas, se tiene que tener de la primera a la última para cada nodo. Las $n_i, \forall i = 1, \dots, m$ de cada derivada no tienen que ser las mismas.

Si contamos el número de condiciones que imponemos en cada punto:

$$\begin{array}{c} Condiciones\ en\ x_0 \rightarrow n_0 + 1 \\ Condiciones\ en\ x_1 \rightarrow n_1 + 1 \\ \vdots \\ Condiciones\ en\ x_i \rightarrow n_i + 1 \end{array} \qquad (33)$$

El número de condiciones totales es por tanto $N = n_0 + n_1 + n_3 + \cdots + n_n + n + 1$.

El grado del polinomio es entonces igual al número de condiciones totales menos 1, es decir,

$$N - 1 = n_0 + n_1 + n_3 + \cdots + n_n + n.$$

Diferencias divididas generalizadas:

$$f[x_0, x_0] = f'(x_0)$$

$$f[x_1, x_0] = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \rightarrow Cuando\ x_1 = x_0, f[x_0, x_0] = f'(x_0)$$

$$f[x_0, x_0, x_0] = \frac{f''(x_0)}{2!} \; porque \; f[x_0, x_1, x_2] = \frac{f''(\xi)}{2!}$$

Siendo $\xi \in (\min(x_0, x_1 x_2), \max(x_0, x_1, x_2))$.

Generalizando a $n+1$ valores iguales de x_0,

$$f[x_0, x_0, \dots, x_0] = \frac{f^{(n)}(x_0)}{n!}.$$

Si el primero y el último son distintos, los demás pueden ser iguales,

$$f[x_0, .., x_n] = \frac{f[x_1, \dots, x_n] - f[x_0, \dots, x_{n-1}]}{x_n - x_0} \quad (34)$$

El error de la interpolación de Hermite viene dado por

$$f(x) - p(x) = \frac{f^{(N)}(\theta)}{N!}(x - x_0)^{n_0} \cdots (x - x_i)^{n_i} \cdots (x - x_m)^{n_m}. \quad (35)$$

Siendo $\theta \in (\min\{x, x_i\}, \max\{x, x_i\})$.

Cada factor $(x - x_i)$ va elevado al número de condiciones sobre x_i , y el número de la derivada y el factorial coinciden con el número de condiciones totales.

Debemos notar que la interpolación de Hermite es una generalización por un lado de la interpolación de Taylor, y por otro de la interpolación de Lagrange.

Funciones cerchas interpoladoras. Splines cúbicos.

Supongamos que tenemos $n+1$ puntos:

$$P_k(x_k, y_k), donde \; y_k = f(x_k), k = 0,1, \dots, n$$

En los cuales se quiere interpolar la función f. Las abscisas no es necesario que sean equidistantes, pero se suponen ordenados, osea,

$$x_0 < x_1 < x_2 < \cdots < x_n$$

En ésta sección trataremos la interpolación polinómica a trozos. La idea es encontrar polinomios cúbicos $q_k(x)$ que interpolen la función f en el subintervalo $[x_k, x_{k+1}]$ con

$k = 0,1, \dots, n-1$.

La función $s(x)$ se llama cúbica a trozos en $[x_0, x_n]$ si existen polinomios cúbicos $q_0(x), q_1(x), \dots, q_{n-1}(x)$ tales que:

$$s(x) = q_k(x) \quad en \; [x_k, x_{k+1}], \qquad para \; k = 0,1, \dots, n-1$$

Para que $s(x)$ interpole a los puntos $P_0, P_1, \dots, P_n$ los $q_k(x)$ han de verificar:

$$\begin{cases} q_k(x_k) = y_k \\ q_k(x_{k+1}) = y_{k+1}, \end{cases} \quad k = 0,1,\dots,n-1 \qquad (36)$$

Lo cual supone $2n$ condiciones. Llamaremos a $s(x)$spline cúbico, o simplemente spline, si los polinomios $q_k(x)$ tienen la misma pendiente y la misma concavidad en los nodos que las unen, osea:

$$\begin{cases} q'_{k-1}(x_k) = q'_k(x_k) \\ q''_k(x_k) = q''_k(x_k), \end{cases} \quad k = 1,2,\dots,n-1 \qquad (37)$$

Lo cual supone $2(n-1)$ condiciones a cumplir. Al tener que verificar las condiciones (36) y (37) se asegura que $s(x)$ tiene su primera y segunda derivadas continuas en $[x_0, x_n]$.En éste caso se dice que $s(x)$ es un spline interpolador para $P_0, P_1, \dots, P_N$.

Si $s(x)$ es cúbica a trozos en el intervalo $[x_0, x_n]$, su derivada segunda $s''(x)$ es lineal en el mismo intervalo e interpola en los puntos $\left(x_k, s''(x_k)\right)$ y $\left(x_{k+1}, s''(x_{k+1})\right)$ en $[x_k, x_{k+1}]$.

Por tanto, $q_k(x)$ es un polinomio de grado uno que interpola en los puntos $\left(x_k, s''(x_k)\right)$y $\left(x_{k+1}, s''(x_{k+1})\right)$:

$$q''_k(x) = s''(x_k)\frac{x - x_{k+1}}{x_k - x_{k+1}} + s''(x_{k+1})\frac{x - x_k}{x_{k+1} - x_k}, \quad para\ k = 0,1,..,n-1$$

Denotando con

$$h_k = x_{k+1} - x_k, \quad k = 0,1,\dots.,n-1$$

Y

$$\sigma_k = s''(x_k), \quad k = 0,1,\dots,n$$

Tenemos:

$$q''_k(x) = \frac{\sigma_k}{h_k}(x_{k+1} - x) + \frac{\sigma_{k+1}}{h_k}(x - x_k), \qquad para\ k = 0,1,\dots,n-1 \qquad (38)$$

Donde $h_k\ y\ \sigma_k$ son constantes ($\sigma_k\ a\ determinar$). Integrando dos veces:

$$q_k(x) = \frac{\sigma_k}{h_k}\frac{(x_{k+1} - x)^3}{6} + \frac{\sigma_{k+1}}{h_k}\frac{(x - x_k)^3}{6} + C_k + D_k x \qquad (39)$$

Donde el término lineal lo podemos escribir como:

$$C_k + D_k x = A_k(x - x_k) + B_k(x_{k+1} - x)$$

Siendo $A_k\ y\ B_k$ constantes arbitrarias, quedando entonces:

$$q_k(x) = \frac{\sigma_k}{h_k}\frac{(x_{k+1} - x)^3}{6} + \frac{\sigma_{k+1}}{h_k}\frac{(x - x_k)^3}{6} + A_k(x - x_k) + B_k(x_{k+1} - x) \qquad (40)$$

Aplicando a (40) las condiciones de (36):

$$y_k = \frac{\sigma_k}{h_k}\frac{h_k^3}{6} + \frac{\sigma_{k+1}}{h_k}\,0 + A_k \cdot 0 + B_k h_k = \frac{\sigma_k}{6}\,h_k^2 + B_k h_k \qquad (41)$$

$$y_{k+1} = \frac{\sigma_{k+1}}{h_k}h_k^3 + A_k h_k = \frac{\sigma_{k+1}}{6}h_k^2 + A_k h_k \qquad (42)$$

Y despejando de aquí $A_k\ y\ B_k$ y sustituyendo en (40) resulta:

$$q_k(x) = \frac{\sigma_k}{6}\left[\frac{(x_{k+1}-x)^3}{h_k} - h_k(x_{k+1}-x)\right] + \frac{\sigma_{k+1}}{6}\left[\frac{(x-x_k)^3}{h_k} - h_k(x-x_k)\right] + y_k\left[\frac{x_{k+1}-x}{h_k}\right] + y_{k+1}\left[\frac{x-x_k}{h_k}\right], para\ k = 0,1,\dots,n-1 \qquad (43)$$

Que es la ecuación del spline $q_k(x)$.

Nos falta aún conocer los valores $\sigma_0, \sigma_1, \dots, \sigma_n$ $(n+1\ incógnitas)$ para lo cual usamos (37); derivando en (43) tenemos:

$$q'_k(x) = \frac{\sigma_k}{6}\left[\frac{-3(x_{k+1}-x)^3}{h_k} + h_k\right] + \frac{\sigma_{k+1}}{6}\left[\frac{3(x_k-x)^2}{h_k} - h_k\right] + \frac{y_{k+1}-y_k}{h_k}$$

Por tanto:

$$q'_k(x_k) = \frac{\sigma_k}{6}(-2h_k) + \frac{\sigma_{k+1}}{6}(-h_k) + \frac{y_{k+1}-y_k}{h_k} \qquad (44)$$

$$q'_k(x_{k+1}) = \frac{\sigma_k}{6}(h_k) + \frac{\sigma_{k+1}}{6}(2h_k) + \frac{y_{k+1}-y_k}{h_k} \qquad (45)$$

Reemplazando k por $k-1$ en (45) para obtener $q'_{k-1}(x_k)$ e igualando a (44) nos da:

$$h_{k-1}\sigma_{k-1} + 2(h_{k-1}+h_k)\sigma_k + h_k\sigma_{k+1} = 6\left(\frac{y_{k+1}-y_k}{h_k} - \frac{y_k-y_{k-1}}{h_{k-1}}\right), \qquad (46)$$

$para\ k = 1,2,..,n-1$

$$h_{k-1}\sigma_{k-1} + 2(h_{k-1}+h_k)\sigma_k + h_k\sigma_{k+1} = 6(f[x_k,x_{k+1}] - f[x_{k-1},x_k]) \qquad (47)$$

$para\ k = 1,2,..,n-1$

Como el índice k varía de 1 $n-1$, se producen $n-1$ ecuaciones lineales con $n+1$ incógnitas $\sigma_0, \sigma_1, \dots, \sigma_n$, lo cual produce un sistema subdeterminado que tiene infinitas soluciones.

Existen varias estrategias para eliminar σ_0 de la primera ecuación y σ_n de la $(n-1)-ésima$ produciendo un sistema triangular de orden $(n-1)$ en las variables $\sigma_1, \sigma_2, \dots, \sigma_{n-1}$.

Alternativa I.

Especificar el valor de $s''(x)$ en los puntos extremos: $\sigma_0 = s''(x_0)$ y $\sigma_n = s''(x_n)$. Si se pone $\sigma_0 = 0, \sigma_n = 0$ se denomina spline cúbico natural.

Alternativa II.

Suponer que $s''(x)$ es constante en los extremos: $\sigma_0 = \sigma_1$ y $\sigma_n = \sigma_{n-1}$.

Alternativa III.

Suponer que $s''(x)$ es lineal cerca de los extremos: $\sigma_0 = \frac{1}{h_1}((h_0+h_1)\sigma_1 - h_0\sigma_2)$ y

$$\sigma_n = \frac{1}{h_n - 2}(-h_{n-1})\sigma_{n-2} + (h_{n-2} + h_{n-1})\sigma_{n-1})$$

Alternativa IV.

Especificar el valor de $s'(x)$ en los puntos extremos:

$$\sigma_0 = \frac{3}{h_0}[\Delta y_0 - s'(x_0)] - \frac{1}{2}\sigma_1 \quad y \quad \sigma_n = \frac{3}{h_{n-1}}[s'(x_n) - \Delta y_{n-1}] - \frac{1}{2}\sigma_{n-1}$$

Si hay que calcular muchas veces $s(z)$ entonces es preferible hacer la sustitución:

$$x_{k+1} - z = (x_{k+1} - x_k) - (z - x_k) = h_k - (z - x_k)$$

En $q_k(z)$ y entonces expresar éste en potencias de $z - x_k$ para obtener:

$$q_k(z) = y_k + \alpha_1(z - x_k) + \alpha_2(z - x_k)^2 + \alpha_3(z - x_k)^3 =$$

$$y_k + (z - x_k)(\alpha_1 + (z - x_k)(\alpha_2 + (z - x_k)\alpha_3))$$

Evaluando con sólo 4 sumas/restas y 3 productos, donde

$$\alpha_1 = f[x_k, x_{k+1}] - \frac{h_k}{6}(\sigma_{k+1} + 2\sigma_k), \alpha_2 = \frac{\sigma_k}{2}, \alpha_3 = \frac{\sigma_{k+1} - \sigma_k}{6h_k}$$

En la forma matricial, el sistema tridiagonal que resulta en el caso de spline cúbico natural es:

$$\begin{bmatrix} 2(h_0 + h_1) & h_1 & \dots & 0 \\ h_1 & 2(h_1 + h_2) & \dots & 0 \\ 0 & h_2 & \cdots & 0 \\ \vdots & \vdots & 2(h_{n-3} + h_{n-2}) & h_{n-2} \\ 0 & 0 & h_{n-2} & 2(h_{n-2} + h_{n-1}) \end{bmatrix} \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \vdots \\ \sigma_{n-1} \end{bmatrix}$$

$$= 6\begin{bmatrix} f[x_1, x_2] - f[x_0, x_1] \\ f[x_2, x_3] - f[x_1, x_2] \\ f[x_3, x_4] - f[x_2, x_3] \\ \vdots \\ f[x_{n-1}, x_n] - f[x_{n-1}, x_{n-2}] \end{bmatrix}$$

Ejemplo de aplicación ámbito naval.

Interpolación de líneas de agua mediante splines cúbicos.

En este apartado vamos a ejecutar un caso particular en ingeniería naval del cálculo de splines cúbicos explicado anteriormente.

Para dicha aplicación hemos elegido un diseño del buque U.S. Navy Combatant DTMB 5415, el cual realizamos, mediante el programa de dibujo asistido Rhinoceros, su plano de formas en el curso de dibujo naval del primer curso académico. Dicho buque está definido por las siguientes características:

- Eslora total=178,090 m
- Eslora entre perpendiculares= 146,00 m
- Calado del proyecto=6,150 m
- Manga de flotación del proyecto=19,070 m

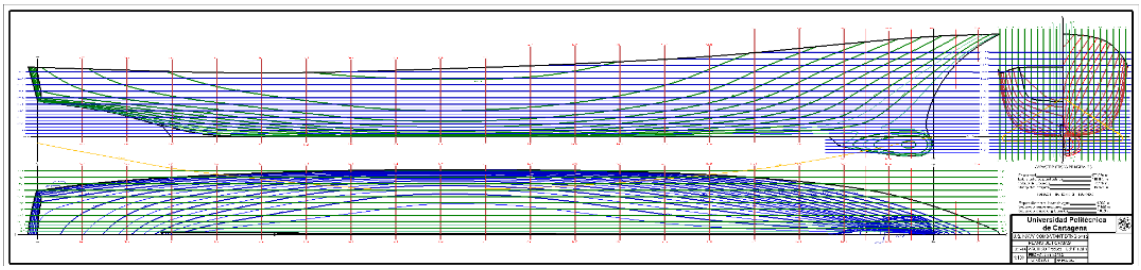


Figura 4.0

Seleccionamos, por ejemplo, la línea de agua número 1 y obtenemos sus nodos o puntos gracias a dicho programa, en su representación en el plano base. Utilizaremos como nodos los correspondientes a las cuadernas en abscisas, separados pues entre sí por la distancia definida como clara de cuadernas. Utilizaremos los puntos comprendidos entre las cuadernas 4 y 17 ambas inclusive, lo que hará un total de 14 nodos.

Midiendo las coordenadas de los distintos puntos en el programa obtenemos los siguientes nodos.

$$[(0.0\ \ 1.64)\ (7.1\ \ 2.89)\ (14.2\ 4.10)\ (21.3\ \ 5.27)\ (28.4\ \ 6.12)\ (35.5\ 6.4)\ (42.6\ \ 6.31)\ (49.7\ 6.89)\ \ldots$$

$$(56.8\ \ 5.14)\ (63.9\ \ 4.17)\ (71\ \ 3.15)\ (78.1\ 2.28)\ (85.2\ 1.64)\ (92.3\ 1.15)]$$

Como condiciones de contorno utilizaremos la descrita en la alternativa uno, según la cual suponemos los valores de las segundas derivadas conocidos e iguales a cero, con lo que así tendremos un spline natural.

$\sigma_0 = 0, \sigma_{13} = 0$.

Para hallar la función spline a trozos utilizamos el programa implementado en la interfaz gráfica. Para ello, una vez abierta la interfaz y seleccionado el método"Splines cúbicos", debemos introducir los datos necesarios.

Para introducir los datos de los nodos que a utilizar para interpolara la fnción deberán ser introducidos por separado los valores de las abscisas y sus correspodientes valores en ordenadas.

Para introducir los valores en abscisas deberá ser pulsado el botón “Crear vector abscisas” dondo el programa llama a un archivo en el que deberán ser introducidos dichos datos, en dicho archivo se detalla el modo de hacerlo.

De igual modo, para introducir los valores de las ordenadas en los nodos que disponemos deberá ser pulsado el botón” Crear vector de ordenadas” que de igual modo nos remite a un archivo a parte donde deberán ser introducidos los valores de las ordenadas en los nodos.

Además, suponemos que la función spline es un spline natural, es decir, que los valores de la segunda derivada de éste en los extremos se anulan. Por último, deseamos evaluar la función en el punto $x = 55\ m$ y que aparezcan cuatro cifras decimales por pantalla.

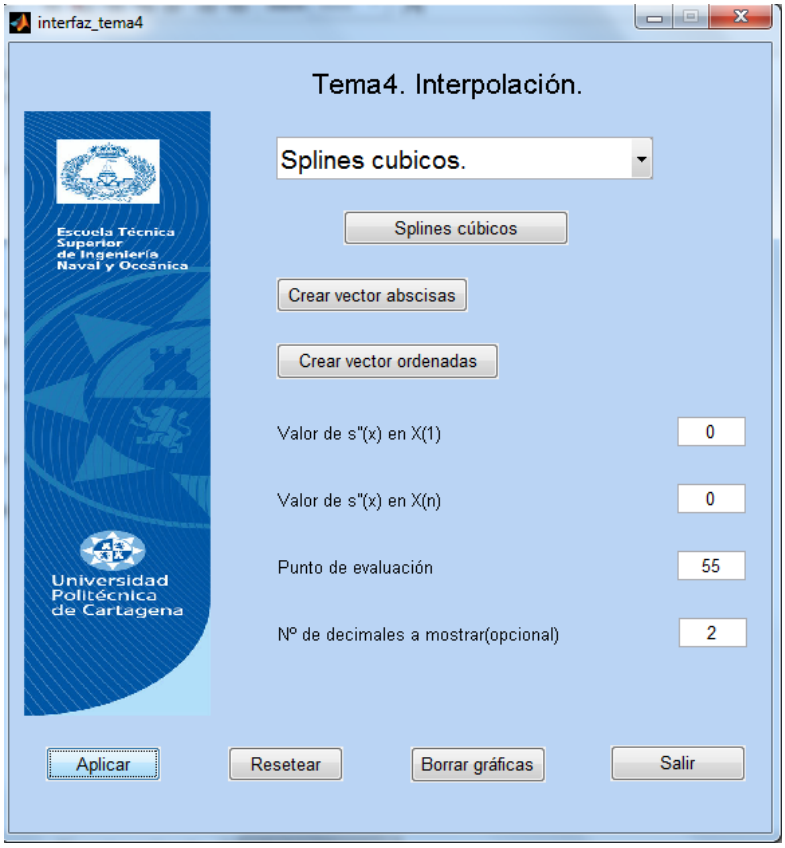


Figura 4.1

Una vez introducidos y guardados los datos necesarios, al hacer click en el botón aplicar aparece un archivo de texto con los resultados obtenidos, además de una gráfica que representa la función spline obtenida,

```
**********************************************************************
*
Vector que contiene las abscisas:
 0.00 7.10 14.20 21.30 28.40 35.50 42.60 49.70 56.80 63.90 71.00 78.10
85.20 92.30
**********************************************************************
*
Vector con la lista de ordenadas:
 1.64 2.89 4.10 5.27 6.12 6.40 6.31 6.89 5.14 4.17 3.15 2.28 1.64 1.15
**********************************************************************
*
Valor de la segunda derivada en X(1):
0.00
**********************************************************************
*
Valor de la segunda derivada en X(n):
0.00
**********************************************************************
*
Punto en el que evaluar la función a trozos:
55.00
**********************************************************************
*
Evaluación del polinomio de Taylor en las abscisas xs:
5.63
**********************************************************************
*
Trozo polinómico que se ha evaluado:
8
**********************************************************************
*

_1.64+0.1778*x-3.374e-005*x^3                    para 0.00< x <7.10
_1.608+0.1915*x-0.001931*x^2+5.693e-005*x^3      para 7.10< x <14.20
_2.326+0.03968*x+0.008758*x^2-0.000194*x^3       para 14.20< x <21.30
_1.063+0.2176*x+0.000406*x^2-6.329e-005*x^3      para 21.30< x <28.40
_5.371-0.2375*x+0.01643*x^2-0.0002514*x^3        para 28.40< x <35.50
_-78.69+6.866*x-0.1837*x^2+0.001628*x^3          para 35.50< x <42.60
_306.4-20.25*x+0.4528*x^2-0.003353*x^3           para 42.60< x <49.70
_-523+29.81*x-0.5544*x^2+0.003403*x^3            para 49.70< x <56.80
_387.9-18.3*x+0.2926*x^2-0.001568*x^3            para 56.80< x <63.90
_-165.1+7.663*x-0.1137*x^2+0.0005512*x^3         para 63.90< x <71.00
_60.07-1.851*x+0.0203*x^2-7.786e-005*x^3         para 71.00< x <78.10
_30.73-0.724*x+0.005876*x^2-1.627e-005*x^3       para 78.10< x <85.20
_70.5-2.124*x+0.02231*x^2-8.056e-005*x^3         para 85.20< x <92.30
```

Figura 4.2

Así ya disponemos de la expresión del polinomio cúbico que interpola a cada uno de los tramos entre los nodos que han sido introducidos, con lo que podremos modificar mejor las formas de nuestra embarcación al disponer de las trayectorias que siguen cada una de las líneas que componen el plano de formas del buque.

Además podemos observar que la representación gráfica de la función spline obtenida se asemeja bastante a la línea de agua que queríamos interpolar, por lo que podemos afirmar que la aproximación ha sido correcta.

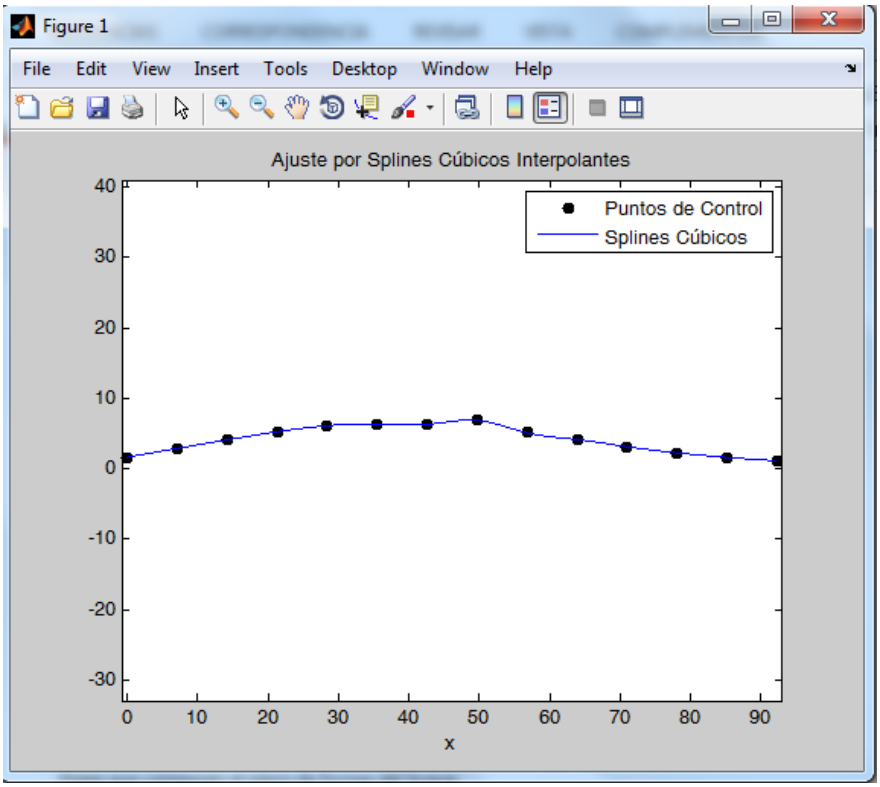


Figura 4.3

Ayuda a la interfaz gráfica

En este apartado explicaremos de forma detallada y concisa los pasos a seguir para poder utilizar y manejar con la habilidad suficiente nuestra interfaz gráfica desarrollada para la mejor comprensión e interiorización de los conocimientos explicados en el tema de interpolación.

Para comenzar, abrimos nuestra interfaz gráfica, del mismo modo que anteriormente ha sido descrito, solo que en este caso seleccionaremos la correspondiente al tema de interpolación, denominada como “Tema4_Interpolación”.

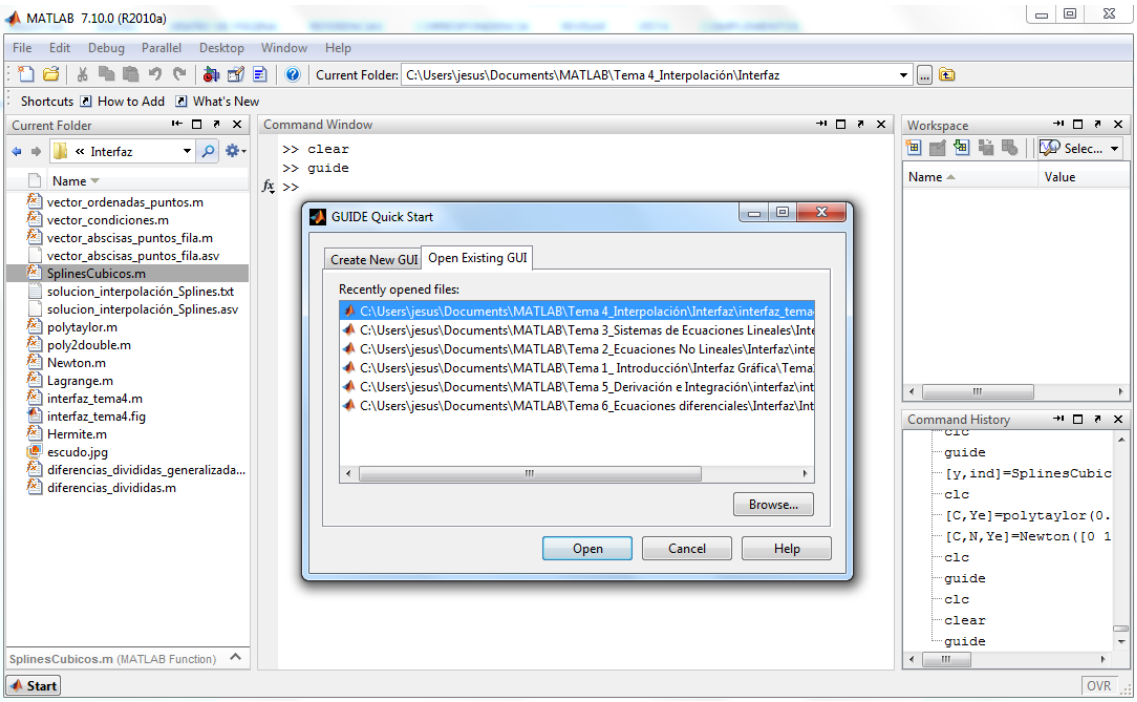


Figura 4.4

Al abrir dicho fichero aparecerá la interfaz de diseño de la propia interfaz gráfica, donde deberemos hacer doble click en el botón verde de reproducir o “play” situado en la zona superior.

Al hacerlo, aparecerá nuestra interfaz gráfica, inicialmente con la lista de métodos que han sido implementados y donde podremos seleccionar el método que queremos reproducir.

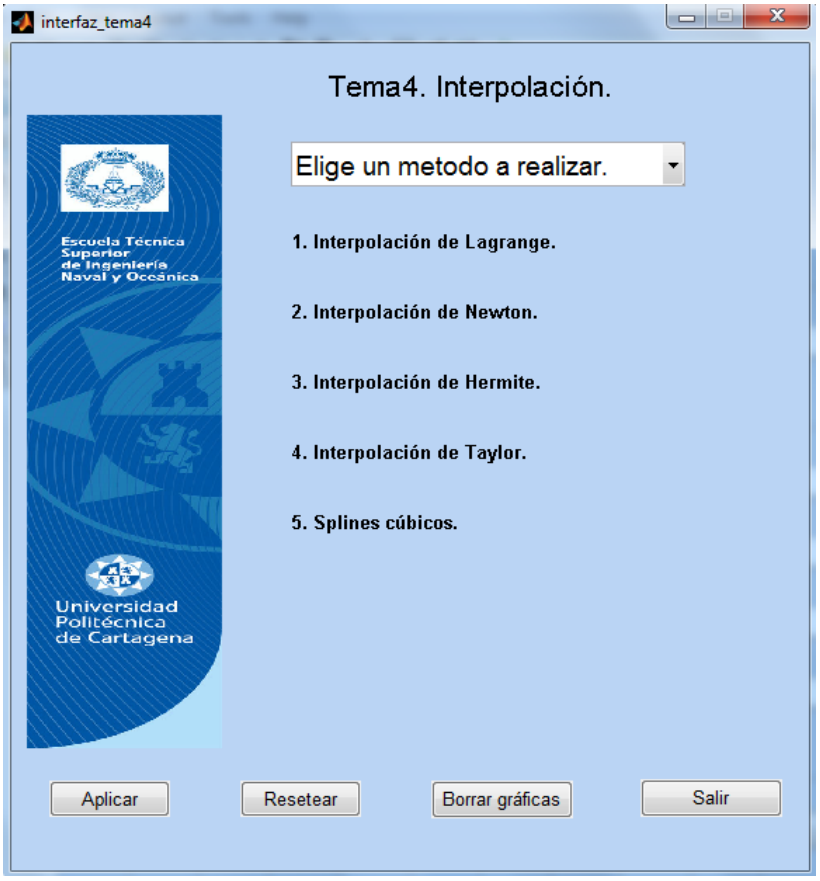


Figura 4.5

En este punto, pulsando en donde dice “Elige un método a realizar” nos aparece automáticamente una lista de los métodos de interpolación, donde elegiremos el que deseemos llevar a cabo.

1. Interpolación de Lagrange.

Una vez seleccionado el método de interpolación de Lagrange nos aparecerá en pantalla el siguiente cuadro.

Figura 4.6

Al hacer doble click en el botón de “Interpolación de Lagrange” el programa nos remitirá a un pdf en el cual se explica la teoría del método de interpolación de Lagrange además de contar con un ejemplo de cómo utilizar la interfaz.

En este ejemplo consideraremos los siguientes puntos a interpolar $(0,-2),(1,6),(3,40)$.

Cuando presionamos el botón de crear vector de abscisas, el programa nos remite a un archivo en el que podremos introducir los datos de nuestro vector que contiene los datos de las abscisas de los puntos, donde además se explica de un modo preciso cómo hacerlo.

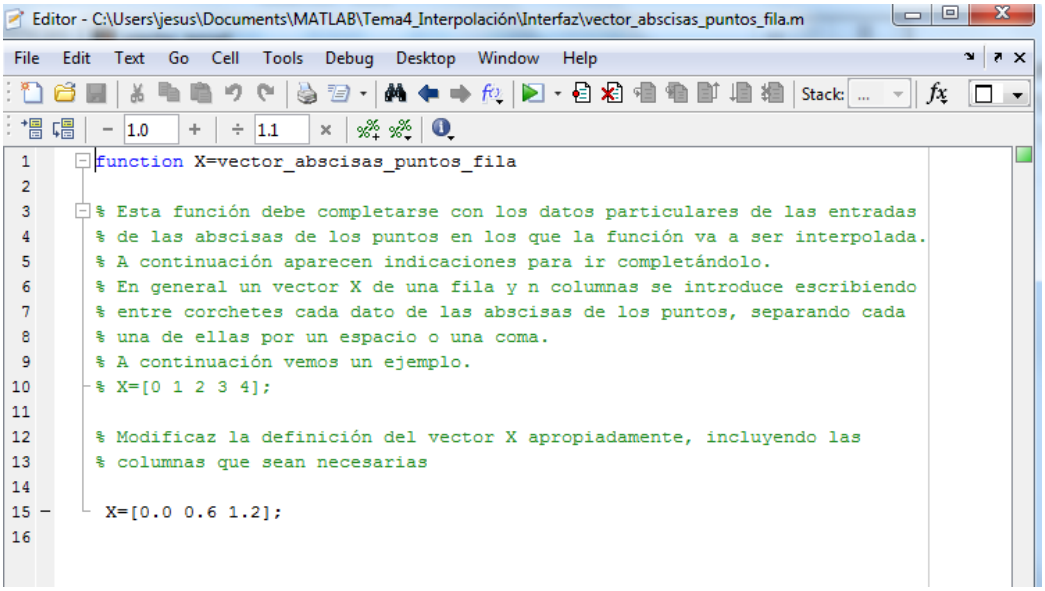


Figura 4.7

Una vez introducido nuestro vector guardamos el archivo.

De igual modo al presionar el botón “crear vector de ordenadas” el programa nos remitirá a un archivo en el que podremos introducir los datos de nuestro vector de ordenadas.

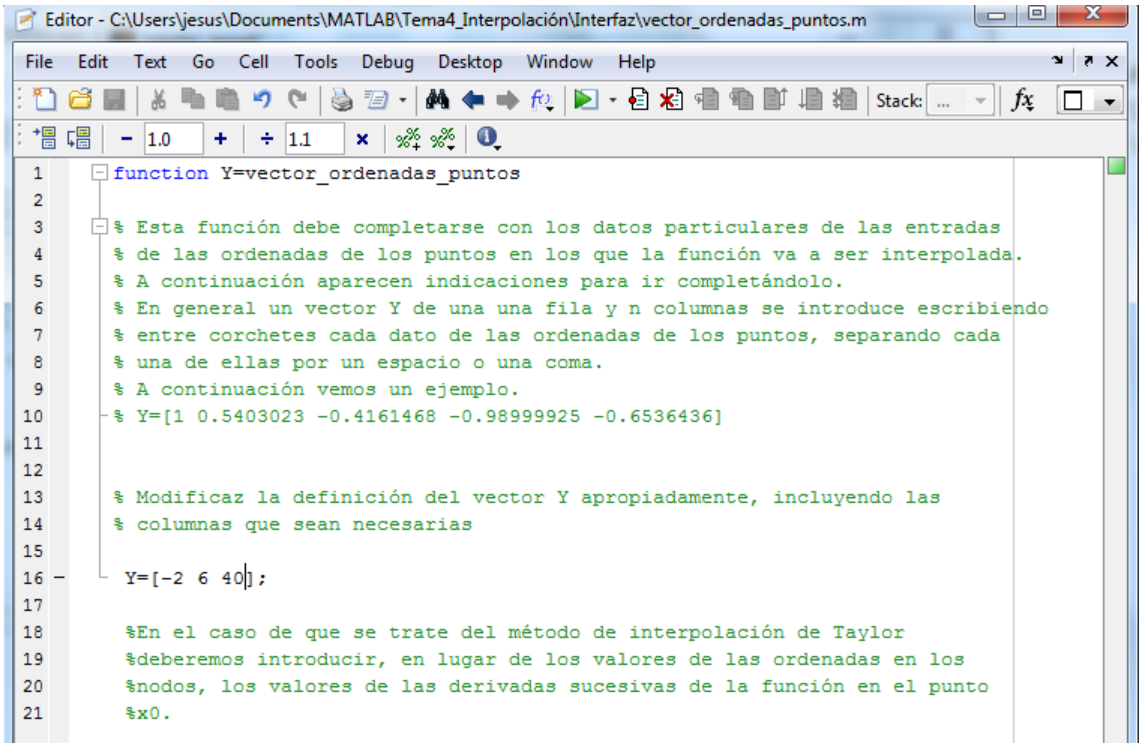


Figura 4.8

Una vez que hemos introducido nuestro vector guardamos el archivo.

A continuación introduciremos, si así lo deseamos, el valor de las abscisas donde queremos evaluar nuestro polinomio obtenido además del número de decimales que queremos mostrar en la pantalla de resultados. Ambos datos pueden ser introducidos de forma solitaria o conjunta.

Figura 4.9

Una vez que tenemos todos los datos necesarios presionamos el botón “Aplicar” apareciendo los siguientes resultados:

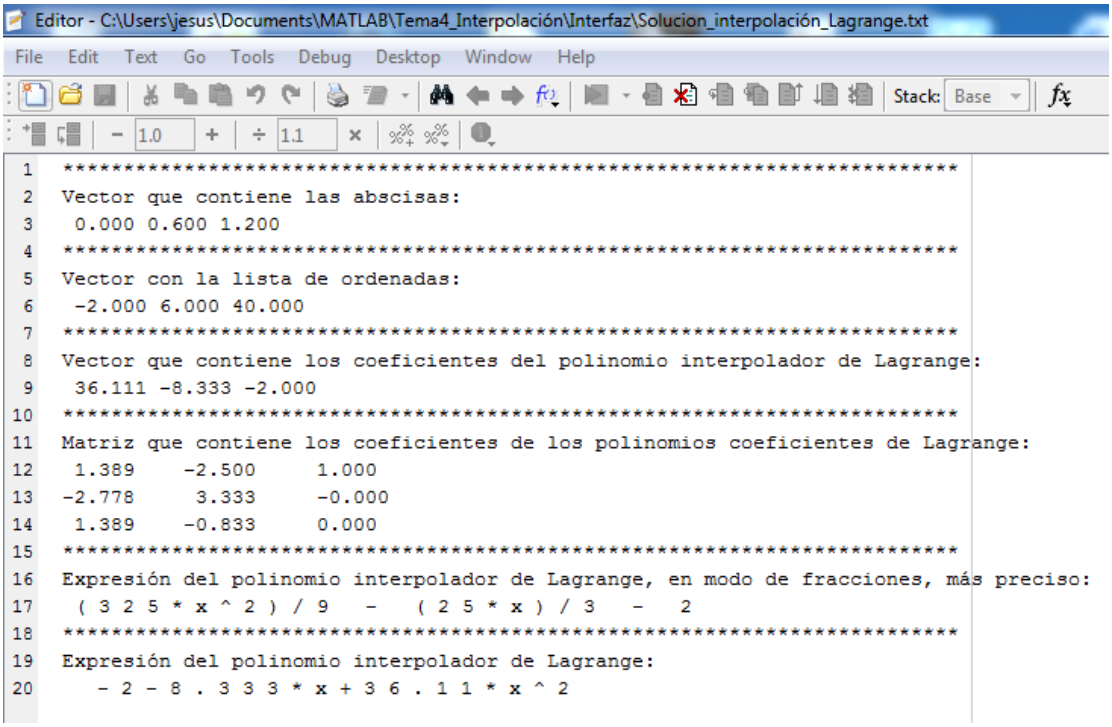


Figura 4.10

Además de la representación gráfica de la función interpoladora obtenida;

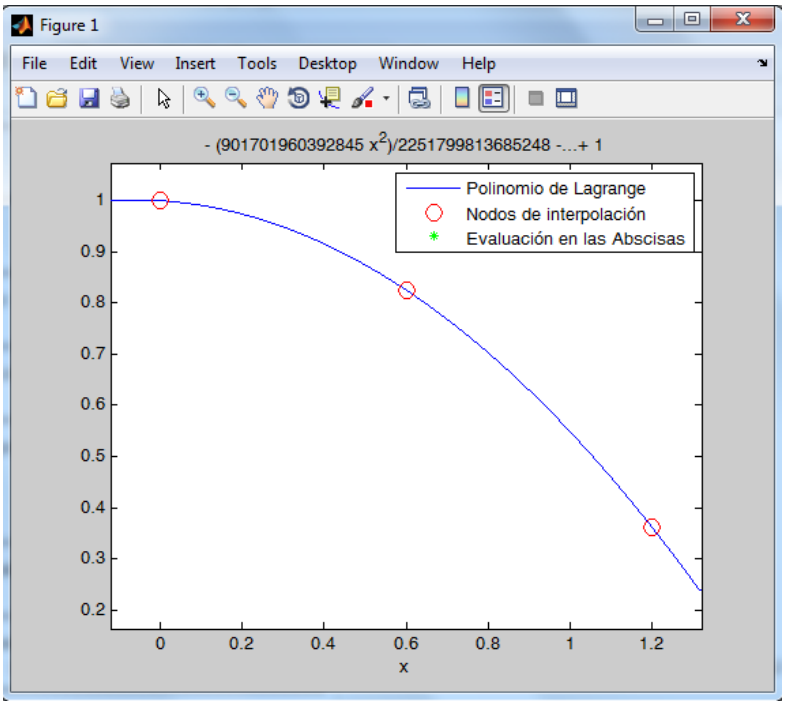


Figura 4.11

2. Método de interpolación de Newton.

Explicaremos ahora cómo realizar el método de interpolación de Newton en nuestra interfaz gráfica, para ello seleccionamos el método, donde los aparecerá el siguiente cuadro:

Figura 4.12

Al igual que en el método de Lagrange anteriormente explicado, al presionar el botón "Interpolación de Newton" el programa nos remitirá a un documento pdf en el que se explica la teoría en la que se basa el método de interpolación de Newton, asi como cómo realizar un ejemplo en nuestra interfaz gráfica.

De igual modo los botones "Crear vector de abscisas" y "Crear vector de ordenadas" nos remiten a archivos en los que introducir los valores de nuestro vector de abscisas y vector de ordenadas, respectivamente.

En este ejemplo, consideraremos los valores de f dados en la tabla siguiente:

x_i	0	1	2	3	4
$f(x_i)$	1	5	31	121	341

Y queremos averiguar el valor de $f(2.5)$ mediante el polinomio de Newton obtenido. Además queremos que aparezcan 3 decimales por pantalla.

Pues bien una vez hemos introducido los valores correspondientes en los vectores de abscisas y ordenadas e introducimos el valor en el que queremos evaluar la función;

Figura 4.13

Entonces al presionar el botón “Aplicar” aparecerá el siguiente archivo de texto con los resultados;

```
*************************************************************************
Vector que contiene las abscisas:
 0.000 1.000 2.000 3.000 4.000
*************************************************************************
Vector con la lista de ordenadas:
 1.000 5.000 31.000 121.000 341.000
*************************************************************************
Punto de abscisas en el que se quiere evaluar el polinomio:
2.500
*************************************************************************
Vector que contiene las diferencias divididas calculadas:
 1.000 4.000 11.000 7.000 1.000
*************************************************************************
Vector que contiene los coeficientes del polinomio interpolador de Newton:
 1.000 1.000 1.000 1.000 1.000
*************************************************************************
Matriz que contiene los coeficientes de los polinomios base para construir el polinomio de Newton:
 0.000     0.000     0.000     0.000     1.000
 0.000     0.000     0.000     1.000     0.000
 0.000     0.000     1.000    -1.000     0.000
 0.000     1.000    -3.000     2.000     0.000
 1.000    -6.000    11.000    -6.000     0.000
*************************************************************************
Evaluación del polinomio de Hermite en las abscisas Xe:
64.438
*************************************************************************
Expresión del polinomio interpolador de Newton,en forma de fraciones, más preciso:
 x ^ 4   +   x ^ 3   +   x ^ 2   +   x   +   1
*************************************************************************
Expresión del polinomio interpolador de Newton:
   1 + 1 * x + 1 * x ^ 2 + 1 * x ^ 3 + 1 * x ^ 4
```

Figura 4.15

Además de la representación gráfica de la función interpoladora obtenida;

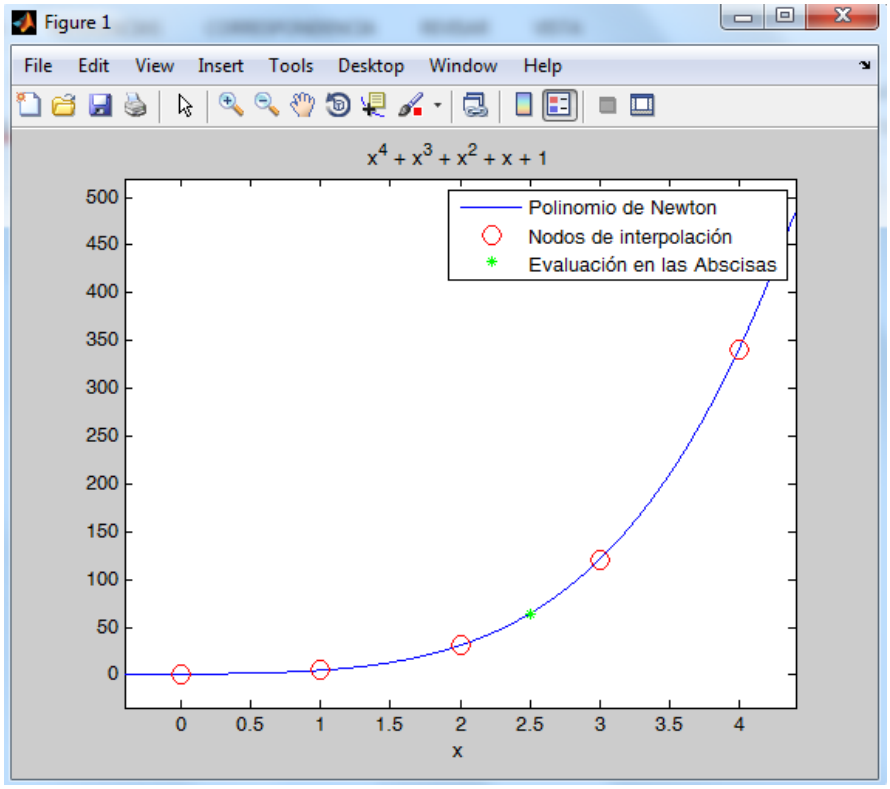


Figura 4.16

3. Método de interpolación de Hermite.

Explicaremos ahora cómo realizar el método de interpolación de Hermite en nuestra interfaz gráfica, para ello seleccionamos el método, donde los aparecerá el siguiente cuadro:

Figura 4.17

Al igual que en los casos anteriores, al presionar el botón “Interpolación de Hermite” el programa nos remitirá a un documento pdf en el que se explica la teoría en la que se basa el método de interpolación de Hermite, así como cómo realizar un ejemplo en nuestra interfaz gráfica.

De igual modo los botones “Crear vector de abscisas” el programa nos remite al archivo donde podremos introducir los valores de nuestro vector de abscisas.

En este caso, sin embargo, el botón “Crear condiciones en cada nodo” nos remite a un archivo, nuevo hasta ahora, en el que podremos introducir los valores de nuestras condiciones en cada nodo. En dicho archivo, al igual que en los demás casos, se explica de forma detallada el proceso a seguir para completar nuestro vector con los datos deseados;

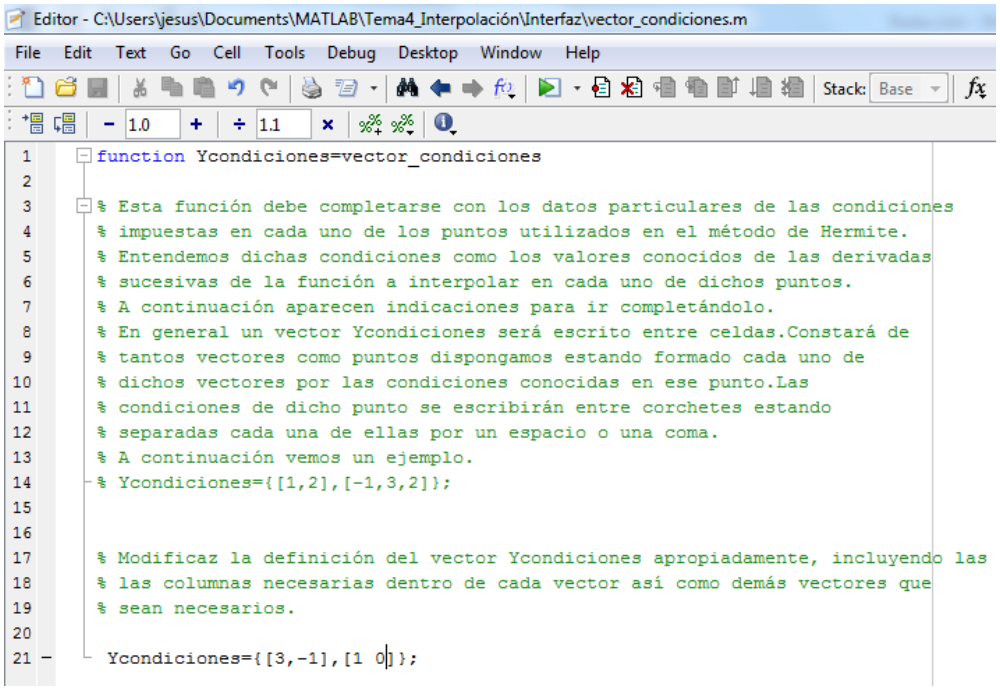


Figura 4.18

El resto de datos como son el punto de evaluación de polinomio obtenido y el número de decimales que deseamos ver en la pantalla de resultados, podremos introducirlos en la interfaz en sus respectivas casillas, si así lo deseamos.

En nuestro ejemplo consideramos un polinomio de grado menor o igual a 3 que satisfaga:

$$p(1) = 3 \quad p(2) = 1$$
$$p'(1) = -1 \quad p'(2) = 0$$

Interpolando, por tanto, entre los puntos $x_0 = 1\ y\ x_1 = 2$

Una vez introducidos los datos de nuestros valores de los vectores de abscisas de los puntos y condiciones en éstos, al pulsar el botón “Aplicar” aparecerá un archivo con la lista de resultados obtenidos;

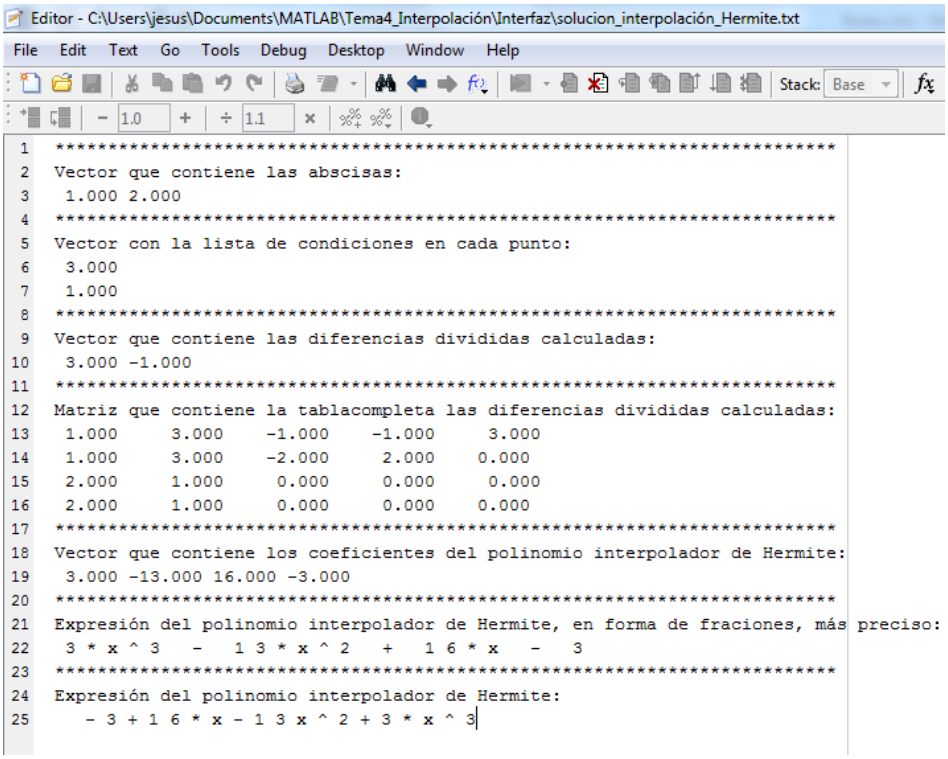


Figura 4.19

Además de la representación gráfica de la función interpoladora obtenida;

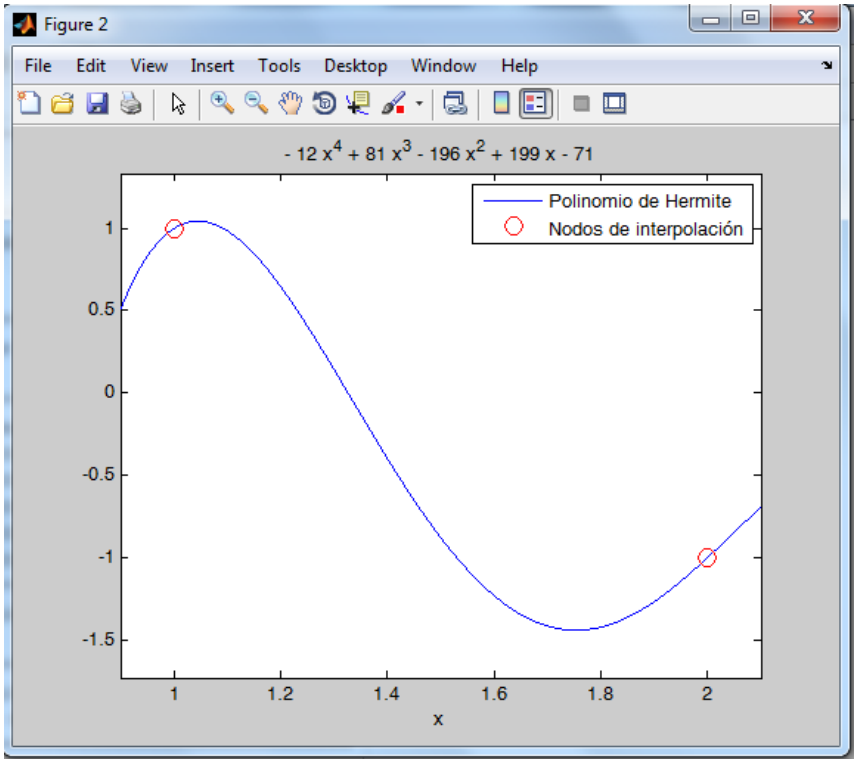


Figura 4.20

4. Método de interpolación de Taylor.

Explicaremos ahora cómo realizar el método de interpolación de Taylor en nuestra interfaz gráfica, para ello seleccionamos el método, donde los aparecerá el siguiente cuadro:

Figura 4.21

Al igual que en los casos anteriores, al presionar el botón “Interpolación de Taylor” el programa nos remitirá a un documento pdf en el que se explica la teoría en la que se basa el método de interpolación de Taylor, así como cómo realizar un ejemplo en nuestra interfaz gráfica.

En este caso, no deberemos crear un vector con los valores de las abscisas de los distintos puntos, sino que desarrollamos nuestro polinomio de interpolación en torno a un único punto x0, cuyo valor en el eje de abscisas introducimos en la casilla correspondiente.

Para poder llevar a cabo el desarrollo del polinomio debemos introducir los valores de las variables sucesivas de la función que queremos interpolar, evaluadas en el punto x0. Para ello pulsamos el botón “crear valores de las derivadas sucesivas” y el programa nos remitirá a un archivo en el que podremos introducir los valores de éstos. En dicho archivo se encuentra explicado de forma precisa el modo en el que debemos introducir dichos valores.

Por otro lado debemos introducir los valores de los extremos del intervalo en el que queremos desarrollar el polinomio, lo cual haremos en modo vector del tipo $[a, b]$.

Por último, podremos introducir el valor del punto donde queremos evaluar el polinomio obtenido así como el número de decimales que queremos mostrar en la pantalla de resultados, si así lo deseamos.

En nuestro ejemplo querremos interpolar la función f entorno al punto $x_0 = 0.5$ cuyos valores de las derivadas sucesivas son;

$$f(0.5) = 0.8776$$
$$f'(0.5) = -0.4794$$
$$f''(0.5) = -0.8776$$
$$f'''(0.5) = 0.4794$$

Además, deseamos interpolar dicha función en el intervalo cerrado $[0,2]$ y evaluar el polinomio obtenido en el punto $x = 1$.

Al introducir estos datos en pantalla nos quedará el siguiente cuadro;

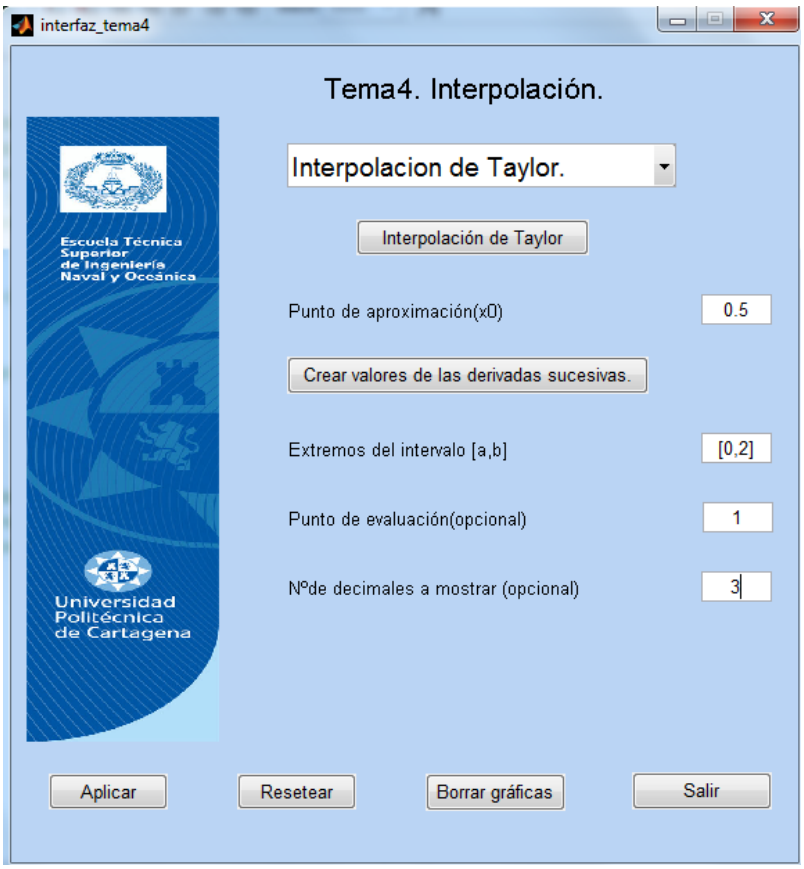


Figura 4.22

Al presionar el botón “Aplicar” nos aparecerá un cuadro de texto con los siguientes resultados,

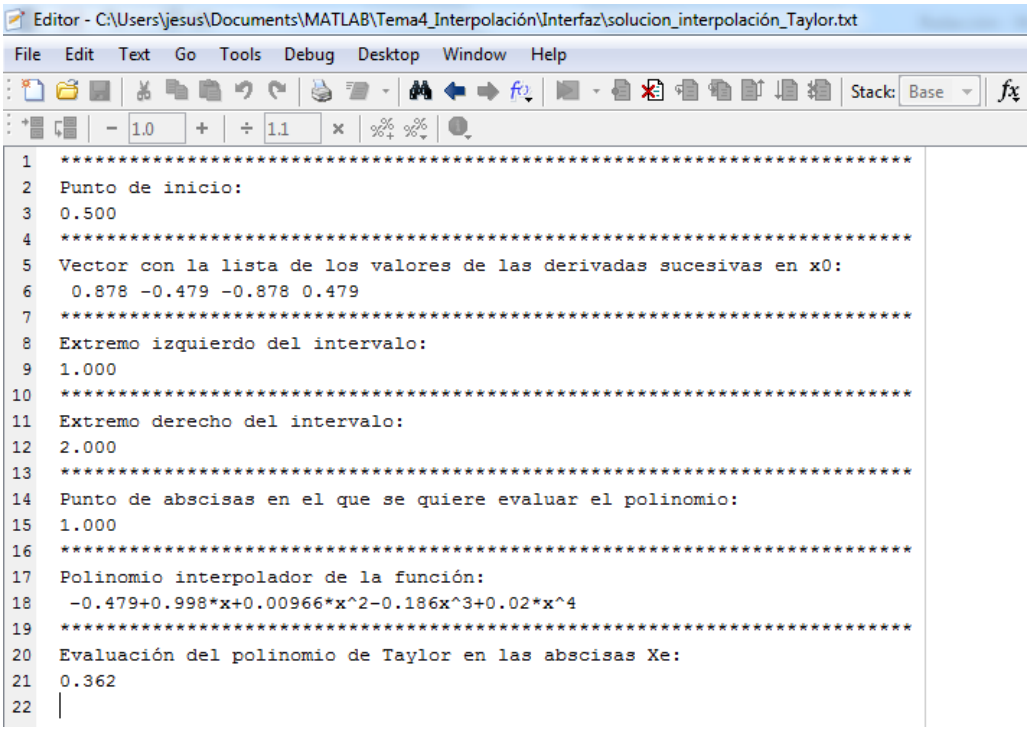


Figura 4.23

Ademas de la gráfica que representa al polinomio interpolador obtenido,

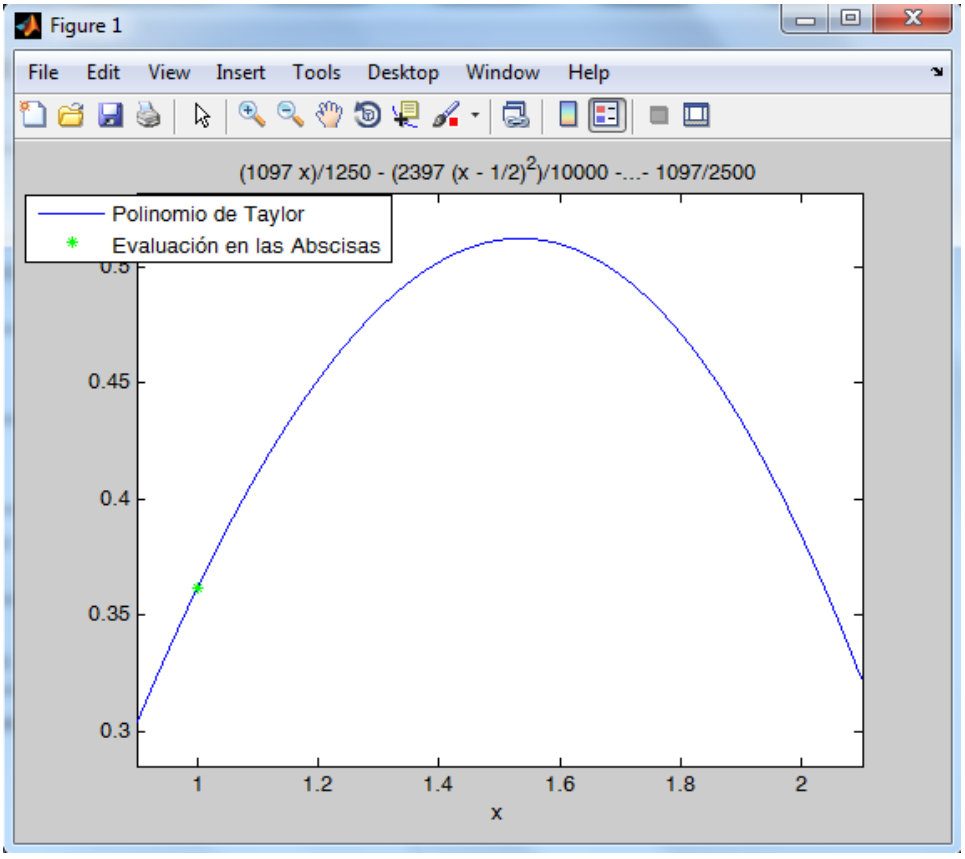


Figura 4.24

5. Splines cúbicos.

Explicaremos ahora cómo realizar el método de interpolación mediante funciones a trozos denominadas splines cúbicos en nuestra interfaz gráfica, para ello seleccionamos el método, donde los aparecerá el siguiente cuadro:

Figura 4.25

Al igual que en los casos anteriores, al presionar el botón “Splines cúbicos” el programa nos remitirá a un documento pdf en el que se explica la teoría en la que se basa el método de interpolación por splines cúbicos, así como cómo realizar un ejemplo en nuestra interfaz gráfica.

En éste programa hemos implementado el método para la alternativa número uno, vista en teoría, según la cual para completar el sistema y que éste sea compatible determinado, suponemos que sabemos los valores de las segundas derivadas en los extremos $s''(x1)$ y $s''(xn)$, si ambos valores son iguales a cero, tenemos lo que se denomina por spline natural.

En este ejemplo interpolamos la siguiente tabla de valores;

0	½	1	2	3	4	5	6	7	7.5	8
0	8.9	11.8	15.5	15.7	15.7	15.7	13.6	9.6	5.8	0

Pues bien, para introducir los valores de las abscisas de nuestros nodos presionamos el botón "Crear vector de abscisas" que nos remitirá a un archivo en el que podremos introducir dichos datos. En dicho archivo se encuentra explicada de forma rigurosa la manera de introducir dichos datos.

De igual forma, al presionar el botón "crear vector de ordenadas" el programa nos remite a otro archivo en el que podremos introducir los valores de las ordenadas de nuestros nodos. En dicho archivo se encuentra explicada de forma detallada la manera de introducir dichos datos.

A continuación, deberemos introducir los valores de $s''(x1)$ y $s''(xn)$ así como del punto en el que queremos evaluar el polinomio, cada uno en sus casillas correspondientes.

Por último, podremos introducir el número de decimales que queremos mostrar por pantalla si así lo deseamos.

Introducimos dichos datos en nuestra interfaz, quedando el siguiente cuadro, con los valores de los nodos introducidos y guardados en los archivos correspondientes;

Figura 4.26

Al presionar el botón aplicar, aparece el siguiente archivo de texto con los resultados,

```
**************************************************************************
*
Vector que contiene las abscisas:
 0.000 0.500 1.000 2.000 3.000 4.000 5.000 6.000 7.000 7.500 8.000
**************************************************************************
*
Vector con la lista de ordenadas:
 0.000 8.900 11.800 15.500 15.700 15.700 15.700 13.600 9.600 5.800 0.000
**************************************************************************
*
Valor de la segunda derivada en X(1):
0.000
**************************************************************************
*
Valor de la segunda derivada en X(n):
0.000
**************************************************************************
*
Punto en el que evaluar la función a trozos:
5.500
**************************************************************************
*
Evaluación del polinomio de Taylor en las abscisas xe:
14.891
**************************************************************************
*
Trozo polinómico que se ha evaluado:
7
**************************************************************************
*

_+20.89*x-12.35x^3                      para 0.000< x <0.500
_-3.261+40.45*x-39.13x^2+13.74*x^3      para 0.500< x <1.000
_12.27-6.155*x+7.476*x^2-1.796x^3       para 1.000< x <2.000
_-12.55+31.08*x-11.14x^2+1.307*x^3      para 2.000< x <3.000
_26.24-7.704*x+1.787*x^2-0.1299x^3      para 3.000< x <4.000
_55.5-29.65*x+7.273*x^2-0.5871x^3       para 4.000< x <5.000
_-65.15+42.74*x-7.204x^2+0.3781*x^3     para 5.000< x <6.000
_173.2-76.45*x+12.66*x^2-0.7254x^3      para 6.000< x <7.000
_560.5-242.4*x+36.37*x^2-1.855x^3       para 7.000< x <7.500
_-1728+673.1*x-85.7x^2+3.571*x^3        para 7.500< x <8.000
```

Figura 4.27

Además de la gráfica representativa de la función interpoladora obtenida,

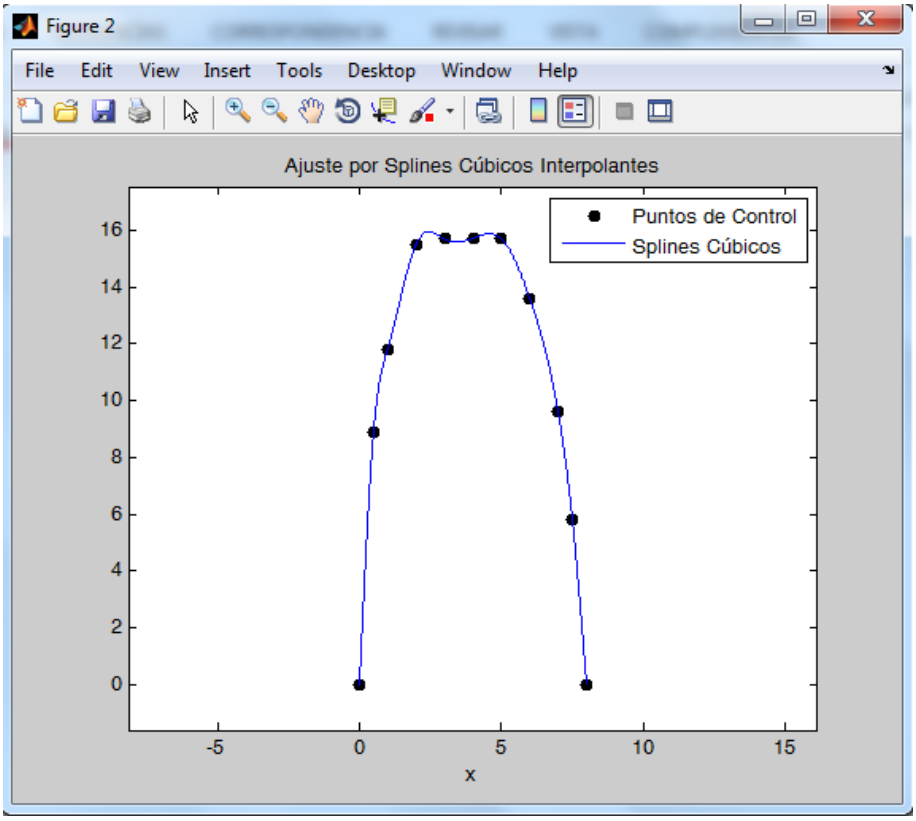


Figura 4.28

Capítulo 5. Derivación e integración numérica.

Las fórmulas de derivación e integración numéricas se emplean cuando se sabe que existen la derivada de una función en un punto y la integral en un intervalo, pero no se sabe calcularlas analíticamente por uno de los motivos que siguen:

- La información que se posee de la función es una tabla de valores,
- La expresión de f es difícil de manejar.

En estos casos, lo habitual es aproximar el valor de $f'(c)$ por una combinación lineal de los valores de f en los puntos $x_i, 0 \leq i \leq n$ en donde está definida, es decir,

$$f'(c) \approx \sum_{i=0}^{n} a_i f(x_i) \qquad (1)$$

Al utilizar la fórmula anterior para dar una aproximación de la derivada estamos cometiendo un error que podemos denotar por $R(f)$ y que se pude definir por la igualdad:

$$f'(c) = \sum_{i=0}^{n} a_i f(x_i) + R(f) \qquad (2)$$

La fórmula (1) se dice que es exacta para ϕ si se verifica que $R(\phi) = 0$.

Uno de nuestros objetivos es el estudio de fórmulas como la (1) y su correspondiente error $R(f)$. La comparación de dos fórmulas de éste estilo puede hacerse de dos formas:

1. Comprobando las expresiones respectivas de $R(f)$.
2. Comparando su exactitud para una familia de funciones derivables en c y definidas por los puntos x_i.

Para el cálculo de integrales seguiremos un proceso análogo: si la integral $\int_a^b f(x)dx$ existe se aproximará por una fórmula de la forma:

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n} a_i f(x_i) \qquad (3)$$

Y el error cometido en dicha aproximación se denotará también por $R(f)$ y vedrá definido por la relación;

$$\int_a^b f(x)dx = \sum_{i=0}^{n} a_i f(x_i) + R(f) \qquad (4)$$

Las fórmulas (1) y (3) se dicen que son exactas de orden r si $R(x_i) = 0$ para todo

$i \in \{0,1,2,..,r\}$.

Derivación numérica.

Introducción.

La principal idea que subyace en las técnicas de derivación numérica está muy vinculada a la interpolación y se podría resumir en lo siguiente: Si de una función f(x) se conocen sus valores en un determinado soporte de puntos, puede "aproximarse" la función f(x) por otra función p(x) que la interpole en dicho soporte y sustituir el valor de las derivadas de f(x) en un punto x por el valor de las correspondientes derivadas de p(x) en dicho punto x.

Una de las primeras fórmulas que nos permiten aproximar una derivada primera tiene sus raíces en los comienzos del cálculo diferencial en el denominado concepto de límite donde la primera derivada de una función f(x) en el punto x se consideraba como el valor del cociente incremental:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

Cuando h disminuye o tiende a cero.

Dos tipos de errores pueden tener lugar cuando aproximamos la derivada de una función en un punto, como ya explicamos en el tema de introducción:

- Error del método o error de truncadura: es debida a sustituir la expresión de la derivada (a través del cálculo de un límite) por una fórmula en la que se combinan valores de la función en determinados puntos y aproximar hasta un cierto orden de la función.
- Error de redondeo que se cometen en las operaciones que contemple la fórmula numérica.

En este tema nos ocuparemos de analizar la primera de las fuentes de error.

Fórmulas de derivación numérica de tipo interpolatorio.

Dada una función real de variable real $f: [a, b] \to R$ para la que conocemos su valor en los puntos (distintos) $\{x_0, x_1, \dots, x_n\}$, es decir, $P(x) = \sum_{i=0}^{n} f_i l_i(x)$ en la forma de Lagrange, para dar la aproximación:

$$f'(c) \approx P'(c) = \sum_{i=0}^{n} f_i l'_i(c) \qquad (5)$$

A la fórmula (5) se le llama de tipo interpolatorio porque se ha obtenido derivando el polinomio de interpolación. Para ésta se tiene que $f(x) = P(x) + E(x)$ siendo $E(x)$ el error de interpolación. Si f es derivable en c entonces el error E también será derivable y se verificará:

$$f'(c) = \sum_{i=0}^{n} f_i l'_i(c) + E'(c) \qquad (6)$$

Siendo el error $R(f) = E'(c)$.

Vemos que llamando $a_i = l'_i(c)$ en la expresión (5) se tiene que ésta fórmula es del tipo (1).

La fórmula $f'(c) \approx \sum_{i=0}^{n} a_i f(x_i)$ es exacta para todo polinomio de grado menor o igual que n si y sólo si es de tipo interpolatorio (o sea si $a_i = l'_i(c_i)$ para $0 \leq i \leq n$ siendo $l_i(x)$ los polinomios de base de Lagrange).

Fórmulas usuales de derivación numérica.

Un punto. Si se conoce un valor único de f en un punto x_0 entonces $f'(c) = 0$.

Dos puntos. Si se conocen $f(x_0) y\, f(x_1)$ entonces el polinomio de interpolación es

$$P(x) = f(x_0) + f[x_0, x_1](x - x_0)$$

Y resulta la fórmula

$$f'(c) \approx f[x_0, x_1] = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \qquad (7)$$

Cuando realizamos la aproximación de la derivada de una función mediante dos puntos, es común utilizar dos puntos que estén separados entre sí una distancia equidistante h. Este proceso lo podemos llevar a cabo de tres formas distintas, según los puntos que elijamos, obteniendo errores distintos en cada caso, tal y como vimos en el tema de introducción.

Dichas formas de realizar esta aproximación son:

1. Progresiva.

   Disponemos de dos puntos $x_0 = c\ y\ x_1 = x + h$. A partir del desarrollo de Taylor:

$$f(c + h) = f(c) + f'(c)h + \frac{h^2}{2} f''(c) + \cdots$$

Se deduce que

$$f'(c) = \frac{f(c + h) - f(c)}{h} - \frac{h}{2} f''(c) + \cdots =$$

$$= \frac{f(c + h) - f(c)}{h} + O(h). \quad (8)$$

Con el término $O(h)$ expresamos que la serie es de convergencia lineal o de primer orden, es decir, que error tiende a cero a la misma velocidad que lo hace h. Este término representa el error cometido en la aproximación y que podemos expresar en la forma;

$$R(f) = -\frac{h}{2} f''(\xi)$$

Siendo ξ un punto intermedio entre $c\ y\ c + h$.

2. Regresiva.

Disponemos de dos puntos $x_0 = c\ y\ x_1 = x - h$. A partir del desarrollo de Taylor:

$$f(c-h) = f(c) - f'(c)h + \frac{h^2}{2}f''(c) + \cdots$$

Se deduce que

$$f'(c) = \frac{f(c) - f(c-h)}{h} + \frac{h}{2}f''(c) + \cdots =$$

$$= \frac{f(c) - f(c-h)}{h} + O(h). \quad (9)$$

Es decir, al igual que en el caso anterior, el error en la aproximación disminuye de forma lineal conforme disminuye h. De igual modo que en el caso anterior, podemos representar dicho error en la forma;

$$R(f) = \frac{h}{2}f''(\xi)$$

Siendo ξ un punto intermedio entre $c\ y\ c + h$.

3. Centrada.

En este caso disponemos de dos puntos situados en torno a c; $x_0 = c + h\ \ y$

$x_1 = c - h$. Realizando los desarrollos de las series de Taylor correspondientes;

$$f(c+h) = f(c) + hf'(c) + \frac{h^2}{2}f''(c) + \frac{h^3}{6}f'''(c) + \cdots$$

$$f(c-h) = f(c) - hf'(c) + \frac{h^2}{2}f''(c) - \frac{h^3}{6}f'''(c) + \cdots$$

Se deduce

$$f'(c) = \frac{f(c+h) - f(c-h)}{2h} - \frac{h^2}{6}f'''(c) + \cdots$$

$$= \frac{f(c+h) - f(c-h)}{2h} + O(h^2) \quad (10)$$

Es de decir que en éste caso el error en la aproximación tiende a cero a la misma velocidad que lo hace h^2, por lo que la aproximación converge más rápidamente a la solución que en los otros dos casos anteriores. En este caso, y suponiendo que f sea de clase C^3en todo el intervalo $[c - h, c + h]$, podemos representar el error de la forma;

$$R(f) = -\frac{h^2}{6}f'''(\xi)$$

Siendo ξ un punto intermedio entre $c\ y\ c + h$.

Tres puntos. Usando interpolación en los tres puntos $x_0, x_1\ y\ x_2$.

Si seleccionamos tres puntos consecutivos separados entre ellos una distancia equiespaciada h, podemos tener a su vez, al igual que el caso anterior, de dos formas distintas según la selección de dichos puntos.

Si suponemos un intervalo cerrado en donde la función f es simétrica

- Pregresiva. Puntos $c, c+h, c+2h.$

En primer lugar, realizamos los desarrollos de la serie de Taylor para cada uno de los puntos;

$f(c)$

$$f(c+h) = f(c) + hf'(c) + \frac{h^2}{2!}f''(c) + \frac{h^3}{3!}f'''(c) + \cdots$$

$$f(c+2h) = f(c) + 2hf'(c) + \frac{(2h)^2}{2!}f''(c) + \frac{(2h)^3}{3!}f'''(c) + \cdots$$

Ahora bien, sabemos que la expresión de la primera derivada tendrá la siguiente forma;

$$f'(c) = \frac{Af(c) + Bf(c+h) + Cf(c+2h)}{h}$$

Con el fin de hallar los coeficientes $A, B\ y\ C$ multiplicamos los desarrollos de la serie de Taylor por estos, y obtenemos el siguientes sistema de ecuaciones lineales con tres ecuaciones y tres incógnitas, donde hemos igualado a cero los términos que queremos que se anulen, y a uno el que queremos obtener, en este caso los términos correspondientes a la primera derivada;

$$\begin{cases} A + B + C = 0 & (f(c)) \\ B + 2C = 1 & (f'(c)) \\ \frac{1}{2}B + 2C = 0 & (f''(c)) \end{cases}$$

Operamos y resolvemos por el método que consideremos, por ejemplo Gauss, y obtenemos los siguientes valores de los coeficientes;

$$A = -\frac{3}{2}, B = 2, C = -\frac{1}{2}$$

Con ellos ya podemos saber la expresión de nuestra primera derivada;

$$f'(c) = \frac{-3f(c) + f(c+h) - f(c+2h)}{2h} + R(f)$$

Para obtener el término del error cometido, debemos buscar la siguiente ecuación linealmente independiente con el resto de ecuaciones, en este caso estará compuesta por los coeficientes que acompañan a los términos de las segundas derivadas, en cada uno de los desarrollos de Taylor;

$$R(f) = \left(\frac{1}{6} * 2 + \frac{8}{6} * (-\frac{1}{2})\right) f'''(\xi)h^3 = -\frac{1}{3}f'''(\xi)h^3$$

Con lo que obtenemos la expresión final de nuestra derivada:

$$f'(c) = \frac{-3f(c) + f(c+h) - f(c+2h)}{2h} - \frac{1}{6}f'''(\xi)h^2 = \cdots$$

$$f'(c) = \frac{-3f(c) + f(c+h) - f(c+2h)}{2h} + O(h^2). \quad (11)$$

- Regresiva. Puntos $c-2h, c-h, c$.

Para obtener la expresión de la primera derivada con puntos regresivos bien podríamos seguir el desarrollo anterior cambiando los puntos en los que desarrollamos las expresiones de Taylor. Sin embargo, al haber ya obtenido la expresión en forma progresiva, podemos realizar un cambio de variable en la función para obtener los coeficientes necesarios. Suponemos que la función es simétrica respecto al punto medio del intervalo, en el intervalo.

Llamamos $s(x) = f(x+c)$

$$g(x) = s(-x) = f(-x+c) \quad \rightarrow \quad g'(x) = -f'(-x+c)$$

$$g'(0) = -f'(c) = \frac{3f(c) - f(c-h) + f(c-2h)}{2h} + O(h^2) \quad (12)$$

Cuatro puntos.

Obtendremos la expresión de la primera derivada en relación a cuatro puntos, centrados en torno a un punto c sin incluir a él mismo. Los puntos son:$c-2h, c-h, c+h, c+2h$.

Al igual que hemos procedido anteriormente, realizamos el desarrollo de la serie de Taylor para cada uno de los puntos:

$$f(c-2h) = f(c) - 2hf'(c) + \frac{(-2h)^2}{2!}f''(c) + \frac{(-2h)^3}{3!}f'''(c) + \frac{(-2h)^4}{4!}f^{iv}(c) + \frac{(-2h)^5}{5!}f^{v}(c) + \cdots$$

$$f(c-h) = f(c) - hf'(c) + \frac{(-h)^2}{2!}f''(c) + \frac{(-h)^3}{3!}f'''(c) + \frac{(-h)^4}{4!}f^{iv}(c) + \frac{(-h)^5}{5!}f^{v}(c) + ..$$

$$f(c+h) = f(c) + hf'(c) + \frac{(h)^2}{2!}f''(c) + \frac{(h)^3}{3!}f'''(c) + \frac{(h)^4}{4!}f^{iv}(c) + \frac{(h)^5}{5!}f^{v}(c) + \cdots f(c)$$

$$f(c+2h) = f(c) + 2hf'(c) + \frac{(2h)^2}{2!}f''(c) + \frac{(2h)^3}{3!}f'''(c) + \frac{(2h)^4}{4!}f^{iv}(c) + \frac{(2h)^5}{5!}f^{v}(c) + ..$$

Como sabemos, la forma de nuestra primera derivada será la siguiente:

$$f'(c) = \frac{Af(c-2h) + Bf(c-h) + Cf(c+h) + Df(c+2h)}{h} + R(f)$$

Pues bien, tal y como hicimos en el caso anterior, multiplicamos cada uno de los desarrollos de Taylor por uno de los coeficientes, y obtenemos el siguiente sistema lineal con tantas ecuaciones como incógnitas:

$$\begin{cases} A + B + C + D = 0 \\ -2A - B + C + 2D = 1 \\ 2A + \frac{1}{2}B + \frac{1}{2}C + 2D = 0 \\ -\frac{4}{3}A - \frac{1}{6}B + \frac{1}{6}C + \frac{4}{3}D = 0 \end{cases}$$

Operando y resolviendo obtenemos los siguientes valores de nuestros coeficientes:

$$A = \frac{1}{12}; B = -\frac{2}{3}; C = \frac{2}{3}; D = -\frac{1}{12}.$$

Entonces la expresión de nuestra primera derivada será:

$$f'(c) = \frac{f(c-2h) - 8f(c-h) + 8f(c+h) - f(c+2h)}{12h} + R(f)$$

Tal y como definimos anteriormente, para hallar la expresión del error $R(f)$ debemos buscar la siguiente ecuación, compuesta por los coeficientes de la derivada sucesiva a la última utilizada en el sistema, que no sea combinación lineal de las anteriores. En este caso, resulta que la ecuación correspondiente a los términos de la cuarta derivada es combinación lineal de las anteriores, por lo que utilizamos la ecuación correspondiente a la quinta derivada, siendo, esta sí, linealmente independiente con las anteriores.

Por ello, nos quedará el término del error como sigue:

$$R(f) = \left(-\frac{32}{120} * \frac{1}{12} + \frac{1}{120} * \frac{2}{3} + \frac{1}{120} * \frac{2}{3} - \frac{32}{120} * \frac{1}{12}\right) f^v(\xi)h^4 = -\frac{1}{30} f^v(\xi)h^4,$$

Con lo que nuestra expresión nos queda finalmente como:

$$f'(c) = \frac{f(c-2h) - 8f(c-h) + 8f(c+h) - f(c+2h)}{12h} - \frac{1}{30} f^v(\xi)h^4 =$$

$$f'(c) = \frac{f(c-2h) - 8f(c-h) + 8f(c+h) - f(c+2h)}{12h} + O(h^4). \quad (13)$$

Es decir que el error cometido en la aproximación, tiende a cero a la misma velocidad que lo hace h^4 , por lo que tendremos una velocidad de convergencia mucho mayor que en los anteriores casos.

Podemos deducir, por tanto, que el error cometido en las aproximaciones disminuye de forma exponencial conforme aumentamos el número de puntos de los que disponemos o tenemos datos.

Además, al seleccionar una sucesión de puntos equiespaciados centrados en torno a un punto, obtenemos un orden de convergencia superior al obtenido en sucesiones regresivas o progresivas.

Fórmulas de derivación numérica de un orden superior.

Estas fórmulas se obtienen sin dificultad, por el mismo procedimiento, derivando k veces el polinomio de interpolación para hallar un valor aproximado de $f^{(k)}(c)$, si bien en este caso el número de puntos a utilizar debe ser mayor que el orden de derivación, pues en otro caso se obtendría $f^{(k)}(c) \approx p^{(k)}(c) = 0$.

Segunda derivada.

Tres puntos.

En el caso particular de que $k = 2$, utilizando tres puntos x_0, x_1, x_2 obtendremos la aproximación

$$f''(c) \cong 2f[x_0, x_1, x_2]$$

Y usando puntos simétricos respecto al central $x_0 = c - h, x_1 = c\ y\ x_2 = c + h$, resulta

$$f''(c) \cong \frac{f(c+h) - 2f(c) + f(c-h)}{h^2} \quad (14)$$

Además si f es de clase $C^{(4)}$ en algún intervalo que contenga a $c - h\ y\ c + h$, se puede probar fácilmente por el desarrollo de Taylor que el error correspondiente adopta la forma

$$R(f) = -\frac{h^2}{12} f^{(4)}(\xi) = O(h^2)$$

Con ξ intermedio entre $c - h\ y\ c + h$.

En este caso se cumple también lo definido anteriormente ya que hemos obtenido un grado de convergencia superior al esperado ya que la sucesión está centrada en torno a un punto c.

Cuatro puntos.

Suponemos un intervalo cerrado en el cual la función f es continua y derivable además de simétrica respecto al punto intermedio.

- Progresiva. Extremo izquierdo del intervalo.

Los puntos a partir de los cuales desarrollaremos la expresión de la segunda derivada son $c, c + h, c + 2h\ y\ c + 3h$. Consideramos que el punto c está en el extremo izquierdo del intervalo. Pues bien, desarrollamos la serie de Taylor correspondiente a cada uno de los puntos:

$f(c)$

$$f(c+h) = f(c) + hf'(c) + \frac{h^2}{2!}f''(c) + \frac{h^3}{3!}f'''(c) + \frac{h^4}{4!}f^{iv}(c) + \cdots$$

$$f(c+2h) = f(c) + 2hf'(c) + \frac{(2h)^2}{2!}f''(c) + \frac{(2h)^3}{3!}f'''(c) + \frac{(2h)^4}{4!}f^{iv}(c) + \cdots$$

$$f(c+3h) = f(c) + 3hf'(c) + \frac{(3h)^2}{2!}f''(c) + \frac{(3h)^3}{3!}f'''(c) + \frac{(3h)^4}{4!}f^{iv}(c) + \cdots$$

Como sabemos, la expresión de la segunda derivada tendrá la forma:

$$f''(c) \approx \frac{Af(c) + Bf(c+h) + C(c+2h) + Df(c+3h)}{h^2}$$

Multiplicamos cada uno de los desarrollos de Taylor por un coeficiente, con el fin de obtener un sistema de ecuaciones lineales que nos permitan obtener dichos coeficientes, igualando en este caso a uno la ecuación formada con los coeficientes de las segundas derivadas:

$$\begin{cases} A + B + C + D = 0 \\ B + 2C + 3D = 0 \\ \frac{B}{2} + 2C + \frac{9}{2}D = 1 \\ \frac{B}{6} + \frac{8}{6}C + \frac{9}{2}D = 0 \end{cases}$$

Operando y resolviendo obtenemos; $A = 2, B = -5, C = 4, D = -1$.

Por lo que tenemos que;

$$f''(c) \approx \frac{2f(c) - 5f(c+h) + 4(c+2h) - f(c+3h)}{h^2}$$

El término del error en este caso lo componen los coeficientes correspondientes a la cuarta derivada, ya que dicha ecuación no es linealmente dependiente con el resto.

$$R(f) = \left(\frac{1}{24} * (-5) + \frac{16}{24} * 4 + \frac{81}{24} * (-1)\right) f^{iv}(\xi)h^2 = -\frac{11}{12} f^{iv}(\xi)h^2$$

Es decir, nuestra expresión para la segunda derivada quedará finalmente como:

$$f''(c) = \frac{2f(c) - 5f(c+h) + 4(c+2h) - f(c+3h)}{h^2} - \frac{11}{12} f^{iv}(\xi)h^2 =$$

$$f''(c) = \frac{2f(c) - 5f(c+h) + 4(c+2h) - f(c+3h)}{h^2} + O(h^2) \qquad (15)$$

- Regresiva. Extremo derecho del intervalo.

Nos centramos ahora en obtener la expresión para la segunda derivada de la función f en el extremo derecho del intervalo, utilizando para ello los puntos $c, c-h, c-2h, c-3h$. Consideramos que el punto c está situado en el extremo derecho del intervalo.

Al haber considerado que la función es continua y derivable y simétrica respecto al punto central dentro del intervalo, podemos obtener la expresión de la segunda derivada en el punto c realizando los cambios de variable apropiados.

Llamamos $s(x) = f(x + c)$

$$g(x) = s(-x) = f(-x + c) \quad \rightarrow \quad \begin{matrix} g'(x) = -f'(-x + c) \\ g''(x) = f''(-x + c) \end{matrix}$$

Es decir los coeficientes no varían en la segunda derivada, por lo que tendremos:

$$f''(c) = \frac{2f(c) - 5f(c-h) + 4(c-2h) - 1f(c-3h)}{h^2} + O(h^2) \quad (16)$$

Cinco puntos.

Realizamos ahora el cálculo de la expresión de la segunda derivada de la función f utilizando cinco puntos que están centrados en torno a un punto c. Tales puntos son $c-2h, c-h, c,\ c+h, c+2h$. Comenzamos desarrollando las series de Taylor para cada uno de los puntos.

$$f(c-2h) = f(c) - 2hf'(c) + \frac{(2h)^2}{2}f''(c) - \frac{(2h)^3}{6}f'''(c) + \frac{(2h)^4}{24}f^{iv}(c) - \frac{(2h)^5}{120}f^{v}(c) + \frac{(2h)^6}{720}f^{vi}(c) + \cdots$$

$$f(c-h) = f(c) - hf'(c) + \frac{(h)^2}{2}f''(c) - \frac{(h)^3}{6}f'''(c) + \frac{(h)^4}{24}f^{iv}(c) - \frac{(h)^5}{120}f^{v}(c) + \frac{(h)^6}{720}f^{vi}(c) + \cdots$$

$f(c)$

$$f(c+h) = f(c) + hf'(c) + \frac{(h)^2}{2}f''(c) + \frac{(h)^3}{6}f'''(c) + \frac{(h)^4}{24}f^{iv}(c) + \frac{(h)^5}{120}f^{v}(c) + \frac{(h)^6}{720}f^{vi}(c) + \cdots$$

$$f(c+2h) = f(c) + 2hf'(c) + \frac{(2h)^2}{2}f''(c) + \frac{(2h)^3}{6}f'''(c) + \frac{(2h)^4}{24}f^{iv}(c) + \frac{(2h)^5}{120}f^{v}(c) + \frac{(2h)^6}{720}f^{vi}(c) + \cdots$$

Por otro lado sabemos que la expresión de la segunda derivada tendrá la siguiente forma:

$$f''(c) = \frac{Af(c-2h) + Bf(c-h) + Cf(c) + Df(c+h) + Ef(c+2h)}{h^2}$$

Por lo que multiplicamos cada una de las expresiones por uno de los coeficientes para obtener un sistema lineal con tantas ecuaciones como incógnitas, en el que hemos igualado a uno los términos correspondientes a la ecuación de la segunda derivada.

$$\begin{cases} A+B+C+D+E=0 \\ -2A-B+D+2E=0 \\ 2A+\frac{1}{2}B+\frac{1}{2}D+2E=1 \\ -\frac{4}{3}A-\frac{1}{6}B+\frac{1}{6}D+\frac{4}{3}E=0 \\ \frac{2}{3}A+\frac{1}{24}B+\frac{1}{24}D+\frac{2}{3}E=0 \end{cases}$$

Simplificando y resolviendo obtenemos;$A=-\frac{1}{12}; B=\frac{4}{3}; C=-\frac{5}{2}; D=\frac{4}{3}; E=-\frac{1}{12}$. Por lo que;

$$f''(c)\approx\frac{-f(c-2h)+16f(c-h)-30f(c)+16f(c+h)-f(c+2h)}{12h^2}$$

El término del error estará compuesto por los coeficientes de la ecuación correspondiente a la derivada sucesiva que no sean combinación lineal con las anteriores. En este caso, la ecuación formada con los términos correspondientes a quinta derivada está en dependencia lineal con las anteriores, no siendo así para la ecuación formada con los términos correspondiente a la sexta derivada. En consiguiente, el término del error será el siguiente:

$$R(f)=\left(\frac{4}{45}*\left(-\frac{1}{12}\right)+\frac{1}{720}*\frac{4}{3}+\frac{1}{720}*\frac{4}{3}+\frac{4}{45}*\left(-\frac{1}{12}\right)\right)f^{iv}(\xi)h^6=-\frac{1}{90}f^{iv}(\xi)h^6,$$

Por lo que nuestra expresión para la segunda derivada será finalmente;

$$f''(c)=\frac{-f(c-2h)+16f(c-h)-30f(c)+16f(c+h)-f(c+2h)}{12h^2}-\frac{1}{90}f^{iv}(\xi)h^4=$$

$$f''(c)=\frac{-f(c-2h)+16f(c-h)-30f(c)+16f(c+h)-f(c+2h)}{12h^2}+O(h^4). \qquad (17)$$

Como podemos observar, en éste caso también hemos obtenido un orden de aproximación mayor del esperado al utilizar cinco puntos, que sería $O(h^3)$ ya que la sucesión de puntos utilizados se encuentra centrada en torno al punto c.

Tercera derivada.

Realizaremos, por último, el desarrollo de la expresión de la tercera derivada para una función f , que consideramos continua y derivable, utilizando para ello cuatro puntos centrado en torno a un punto c, excluyendo a éste. Dichos puntos son $c-2h, c-h, c+h, c+2h$.

Como ya hemos realizado en repetidas ocasiones, desarrollamos las expresiones de las series de Taylor para cada uno de los puntos.

Sabemos pues que la expresión para la tercera derivada utilizando esos puntos tendrá la forma:

$$f'''(c) \approx \frac{Af(c-2h)+Bf(c-h)+Cf(c+h)+Df(c+2h)}{h^3}$$

Pues bien, en el sistema de ecuaciones lineales a resolver para hallar los coeficientes necesarios hemos igualado a uno la ecuación correspondiente a los términos de la tercera derivada;

$$\begin{cases} A+B+C+D=0 \\ -2A-B+C+2D=0 \\ 2A+\frac{B}{2}+\frac{C}{2}+2D=0 \\ -\frac{4}{3}A-\frac{1}{6}B+\frac{1}{6}C+\frac{4}{3}D=1 \end{cases}$$

Simplificando y resolviendo obtenemos; $A=-2; B=4; C=-2; D=2$. Por lo que la expresión de la tercera derivada será;

$$f'''(c) \approx \frac{-f(c-2h)+2f(c-h)-2f(c+h)+f(c+2h)}{2h^3}$$

La ecuación para la derivada $f^{iv}(c)$ es

$\frac{2}{3}A+\frac{1}{16}B+\frac{1}{16}C+\frac{2}{3}D=0$, que es combinación lineal de las anteriores. La siguiente ecuación para $f^{v}(c)$ es

$-\frac{4}{15}A-\frac{1}{120}B+\frac{1}{120}C+\frac{4}{15}D=0$ que no es combinación lineal de las anteriores.

Por tanto el error cometido tendrá la expresión;

$$R(f)=\left(-\frac{4}{15}*\left(-\frac{1}{2}\right)-\frac{1}{120}+\frac{1}{120}(-1)+\frac{4}{15}*\frac{1}{2}\right)f^{v}(\xi)h^5=\frac{1}{4}f^{v}(\xi)h^5$$

Por lo que la expresión final de la tercera derivada será;

$$f'''(c)=\frac{-f(c-2h)+2f(c-h)-2f(c+h)+f(c+2h)}{2h^3}-\frac{1}{4}f^{v}(\xi)h^2=$$

$$f'''(c)=\frac{-f(c-2h)+2f(c-h)-2f(c+h)+f(c+2h)}{2h^3}+O(h^2)\ (18).$$

Integración numérica.

Introducción.

En muchas ocasiones es necesario disponer del valor de la integral de una función f en un intervalo $[a, b]$ pero no nos es posible calcularla por disponer tan sólo de una tabla de valores de la función en cuestión o bien por ser su expresión analítica inmanejable. Entonces, lo más usual es aproximarla por una combinación lineal de los valores de f en los puntos $x_i, i = 0,1, \dots, n$, en los que f es conocida, es decir

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n} a_i f(x_i) \quad (19)$$

Al tomar este valor como aproximación de la integral se comete un error que denotamos por $R(f)$, que está definido por medio de la fórmula

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n} a_i f(x_i) + R(f)$$

Nuestro objetivo en este punto será el estudio de fórmulas de integración (o cuadratura) numérica de este tipo y de los errores correspondientes.

Fórmulas de integración numérica de tipo interpolatorio.

Las fórmulas de integración numérica de tipo interpolatorio son las que se obtienen integrando el polinomio de interpolación. En concreto si se conoce f en $x_i, 0 \leq i \leq n$, entonces:

$$f(x) = P(x) + E(x),$$

Siendo $P(x)$ el polinomio de interpolación de f en los puntos conocidos y $E(x)$ el error de interpolación correspondiente. Si f es integrable en $[a, b]$ se tiene,

$$\int_a^b f(x)dx = \int_a^b P(x)dx + \int_a^b E(x)dx \ (20)$$

Y puesto que $P(x) = \sum_{i=0}^{n} f(x_i) l_i(x)$ resulta:

$$\int_a^b f(x)dx = \sum_{i=0}^{n} \left(\int_a^b l_i(x)dx \right) f(x_i) + R(f)$$

Definiendo $a_i = \int_a^b l_i(x)dx$ nos queda como la fórmula (3). Al igual que en el aparatado anterior, se tiene el siguiente teorema:

La fórmula $\int_a^b f(x)dx \approx \sum_{i=0}^{n} a_i f(x_i)$ es exacta para todo polinomio de grado menor o igual a n si y sólo si es de tipo interpolatorio (o sea si $a_i = \int_a^b l_i(x)\,dx$ para todo $i \in \{0,1, \dots, n\}$, siendo los $l_i(x)$ los polinomios de base de Lagrange).

Fórmulas de tipo interpolatorio usuales.

Un punto. En este caso la fórmula de interpolación numérica usando el punto x_0 se reduce a:

$$\int_a^b f(x)dx \approx f(x_0)(b-a)$$

Y en este caso el error de integración numérica se puede expresar con la relación:

$$R(f) = \int_a^b f[x_0, x](x - x_0)dx$$

En particular, si $x_0 = a$ se tiene la **fórmula del rectángulo**.

$$\int_a^b f(x)dx \approx f(a)(b-a) \ (21)$$

Y si f es de clase C^1 en el intervalo $[a, b]$ el error de integración será:

$$R(f) = f'(\xi)\frac{(b-a)^2}{2}$$

Donde $\xi \in (a, b)$.

En tanto que si $x_0 = \frac{a+b}{2}$ se tiene la **fórmula del punto medio**.

$$\int_a^b f(x)dx \approx f\left(\frac{a+b}{2}\right)(a-b) \ (22)$$

Y si f es de clase C^2 en el intervalo $[a, b]$ el error de integración será

$$R(f) = f''(\xi)\frac{(b-a)^3}{24}$$

Donde $\xi \in (a, b)$

Dos puntos.

Cuando se conoce el valor de f en dos puntos $x_0 \ y \ x_1$ se tiene para el polinomio de interpolación de f en dichos puntos la expresión

$$P(x) = f(x_0) + f[x_0, x_1](x - x_1)$$

Y la integración de éste polinomio dará:

$$\int_a^b P(x)dx = f[x_0](b-a) + f[x_0, x_1]\left(\frac{(b-x_0)^2 - (a-x_0)^2}{2}\right)$$

En particular si tomamos $x_0 = a \ y \ x_1 = b$ obtenemos la **fórmula del trapecio**:

$$\int_a^b f(x)dx \approx \frac{(b-a)}{2}[f(a) + f(b)] \ \ (23)$$

Y si f es de clase C^2 en el intervalo $[a,b]$ el error de integración será

$$R(f) = -f''(\xi)\frac{(b-a)^3}{12}$$

Con $\xi \in (a,b)$.

Fórmulas de $Newton - C\hat{o}tes$ simples.

Son aquellas fórmulas de integración numérica de tipo interpolatorio en las que los puntos de interpolación son equidistantes y dividen al intervalo $[a,b]$ en partes iguales.

Dado un número $n \in N$ y definiendo $h = \frac{b-a}{n}$, podemos definir los puntos en los que interpolaremos como:

$$\begin{cases} x_0 = a \\ x_i = x_0 + ih, \quad 0 \leq i \leq n. \end{cases}$$

Estas fórmulas en las que se incluyen los puntos extremos de los intervalos se llaman fórmulas cerradas, aquellas en las que no se incluyen los extremos se llaman abiertas. A continuación vemos fórmulas cerradas para tres, cuatro y cinco puntos de interpolación.

Tres puntos.

En este caso interpolaremos en los puntos $a, \frac{a+b}{2}, b$ y la fórmula resultante se llama fórmula de Simpson (una de las más utilizadas por su simplicidad y precisión):

$$\int_a^b f(x)dx \approx \frac{b-a}{6}[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)] \quad (24)$$

El error correspondiente a esta fórmula si f es de clase C^4en $[a,b]$:

$$R(f) = -f^{(4)}(\xi)\frac{h^5}{90} = -f^{(4)}(\xi)\frac{(b-a)^5}{2880}$$

Donde $\xi \in (a,b)$ y en este caso $h = \frac{b-a}{2}$.

De esta expresión del error se deduce que la fórmula de Simpson es exacta si el polinomio es de grado menor o igual que tres.

Cuatro puntos.

En este caso interpolaremos en los puntos $a, \frac{2a+b}{3}, \frac{a+2b}{3}, b$ y la fórmula resultante es:

$$\int_a^b f(x)dx \approx \frac{b-a}{8}\left[f(a) + 3f\left(\frac{2a+b}{3}\right) + 3f\left(\frac{a+2b}{3}\right) + f(b)\right] \quad (25)$$

El error correspondiente a esta fórmula si f es de clase C^4en $[a,b]$:

$$R(f) = -f^{(4)}(\xi)\frac{3h^5}{80}$$

Donde $\xi \in (a,b)$ y en este caso $h = \frac{b-a}{3}$.

Cinco puntos.

En este caso interpolaremos en los puntos $x_0 = a, x_1 = a+h, x_2 = a+2h, x_3 = a+3h, x_4 = a+4h = b$ y la fórmula resultante es:

$$\int_a^b f(x)dx \approx \frac{b-a}{90}(7f_0 + 32f_1 + 12f_2 + 32f_3 + 7f_4) \quad (26)$$

El error correspondiente a esta fórmula si f es de clase C^6en $[a,b]$:

$$R(f) = -f^{(6)}(\xi)\frac{8h^7}{945}$$

Donde $\xi \in (a,b)$.

En las fórmulas de Newton-Cotes cerradas con n par $(n+1\ \ nodos)$, el error es un infinitésimo del orden de h^{n+3}, por el contrario, para n impar el error es del orden de h^{n+2}(De aquí la ventaja de utilizar fórmulas como la de Simpson, con n par).

Fórmulas de cuadratura compuestas.

Son las que se obtienen al dividir el intervalo de integración $[a,b]$ en n subintervalos iguales y a cada uno de estos aplicarles una fórmula sencilla, por ejemplo la fórmula del trapecio o la regla de Simpson, estas fórmulas tienen buenas propiedades de estabilidad y convergencia. Veamos a modo de ejemplo las fórmulas compuestas del trapecio y de Simpson.

La regla del trapecio compuesta.

El error de la regla del trapecio puede reducirse si utilizamos la regla del trapecio compuesta, ésta consiste en reducir el intervalo $[a,b]$ en n subintervalos de longitud $h = \frac{b-a}{n}$ y aplicar la regla del trapecio simple a cada uno de los intervalos $[a+jh, a+(j+1)h]\ con\ j \in \{0,1,..,n-1\}$.

El método proporciona la aproximación:

$$\int_a^b f(x)dx \approx \frac{h}{2}\left(f(a) + 2\sum_{i=1}^{n-1} f(a+ih) + f(b)\right) \quad (27)$$

Para esta aproximación, el error que se comete si f es de clase C^2 es:

$$R(f) = -\frac{1}{12}(b-a)h^2 f''(c)$$

Siendo c un punto del intervalo (a, b).

La regla de Simpson compuesta.

Al igual que en la regla del trapecio, si subdividimos el intervalo $[a, b]$ en n partes (con n número par) y aplicamos a cada una de ellas la regla de Simpson, se obtiene una mejor aproximación de la integral.

$$\int_a^b f(x)dx \approx$$

$$\frac{h}{3}\left(f(a) + 2\sum_{i=2}^{n/2} f(a + 2(i-1)h) + 4\sum_{i=1}^{n/2} f(a + (2i-1)h) + f(b)\right), \qquad (28)$$

Con un error, si f es de clase C^4, dado por:

$$R(f) = -\frac{1}{180}(b-a)h^4 f^{iv}(c).$$

Estando c en (a, b).

Estabilidad y convergencia.

Como se ha dicho anteriormente, la estabilidad de un método numérico mide su sensibilidad frente a errores de cálculo o en los datos.

Seguidamente damos algunos resultados sobre estabilidad y convergencia.

- Las fórmulas de Newton-Cotes simples son inestables (luego no son recomendables para valores grandes de n).
- Una condición necesaria y suficiente para que un método de cuadratura de tipo interpolatorio sea convergente sobre $C([a, b])$ es que sea estable.
- Las fórmulas del trapecio y Simpson compuestas son estables y convergentes; en general, las fórmulas compuestas de Newton-Cotes, de grado de exactitud r en $n + 1$ puntos, tales que los coeficientes de la correspondiente fórmula simple son positivos son estables.

Ejemplo de aplicación ámbito naval.

La regla de Simpson Compuesta.

El método de integración por Simpson resulta de gran utilidad en problemas relacionados con la arquitectura naval ya que a la hora de realizar muchos cálculos, como pueden ser el cálculo del volumen de agua que desplaza un buque o la situación del punto de aplicación del empuje, ect, necesitamos saber el resultado de la integral de un conjunto de funciones para las cuales solo sabemos los datos de los valores de los puntos a través de la cartilla de trazado del buque. Es decir, podemos obtener los alores de la función en una serie de puntos equiespaciados pero no así la expresión de dicha función.

A modo de ejemplo realizaremos el siguiente ejercicio.

Un buque de $L_{PP} = 120\ m$ y $B = 22{,}5$ tiene un calado de $7{,}2\ m$. Para esa flotación la curva de áreas de secciones viene definida por:

Sec.	0	1	2	3	4	5	6	7	8
Área(m^2)	6,86	30,32	56,50	79,38	98,87	117,674	132,73	142,99	149,86

9	10	11	12	13	14	15	16	17	18
152,25	153,70	152,38	145,50	135,66	119,67	97,47	74,47	53,51	35,61

19	20
22,69	13,28

Nos pide hallar el volumen de trazado.

Para ello debemos integrar el área de las secciones a lo largo de la eslora.

$$\nabla = \int_L A_s dx$$

Para realizar la integral a lo largo de la eslora, en primer lugar abrimos la interfaz gráfica y seleccionamos el método de integración por Simpson. Una vez en ella se selecciona la opción valores de la función y pulsamos el botón “crear valores función”. Este nos remite a un archivo a parte en el que deberán ser introducidos los valores de la función en las ordenadas.

Por otro lado debemos introducir el equiespaciado entre punto, que en este caso será $h = 6\ m$ y los extremos del intervalo, en este caso $[0\ ,120]$.

Figura 5.0

Una vez introducidos y guardados los datos necesarios, al pulsar el botón aplicar aparece una ventana emergente con los resultados obtenidos;

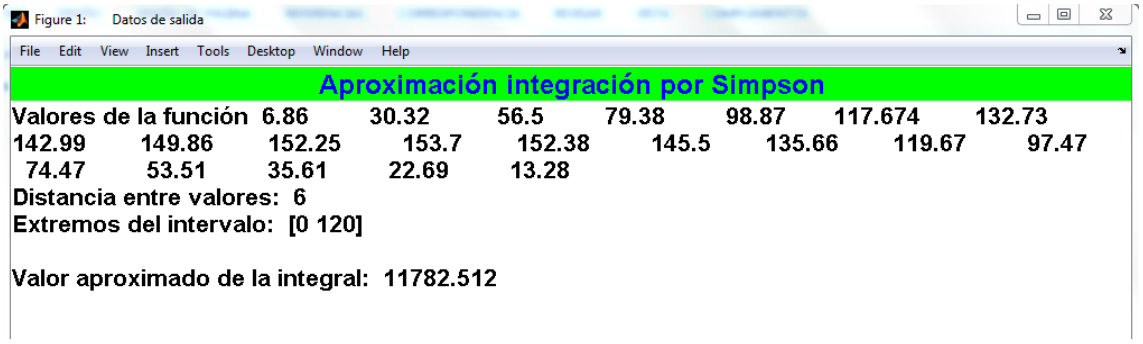


Figura 5.1

Por tanto, el volumen de trazado de nuestro buque al calado de trazado es

$$\nabla = 11782{,}512\ m^3.$$

Ayuda a la interfaz gráfica.

En este apartado explicaremos de forma detallada y concisa los pasos a seguir para poder utilizar y manejar con la habilidad suficiente nuestra interfaz gráfica desarrollada para la mejor comprensión e interiorización de los conocimientos explicados en el tema de interpolación.

Para comenzar, abrimos nuestra interfaz gráfica, del mismo modo que anteriormente ha sido descrito, solo que en este caso seleccionaremos la correspondiente al tema de interpolación, denominada como “Tema5_Derivación e Integración”.

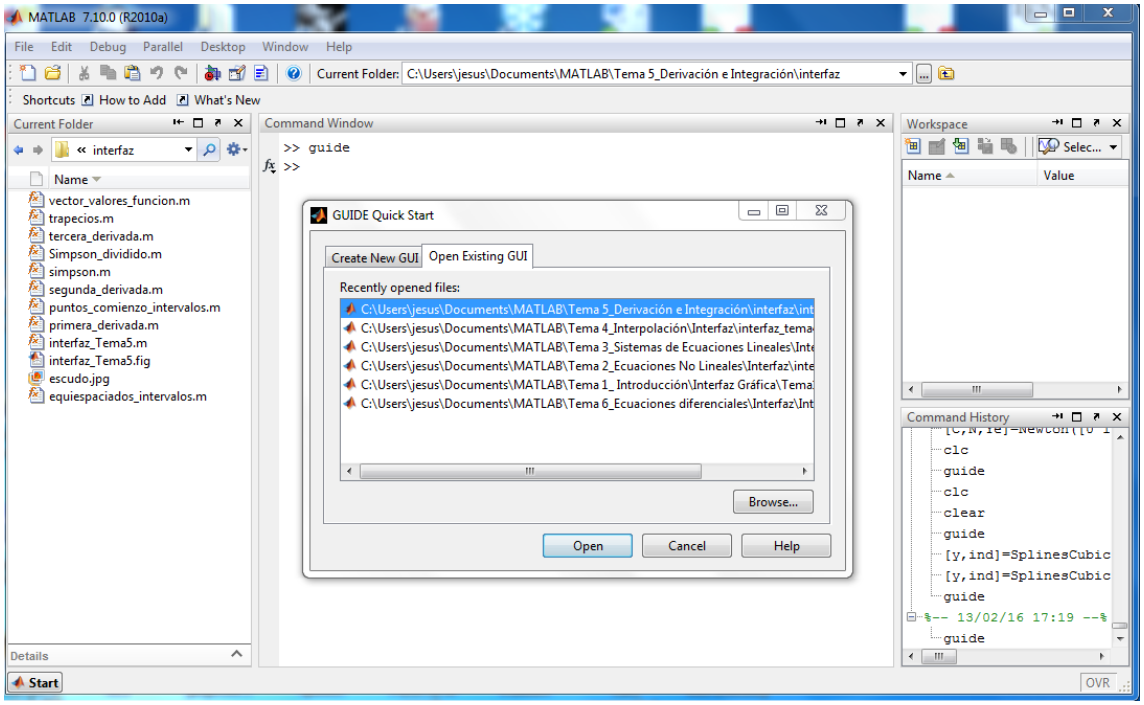


Figura 5.2

Al abrir dicho fichero aparecerá la interfaz de diseño de la propia interfaz gráfica, donde deberemos hacer doble click en el botón verde de reproducir o “play” situado en la zona superior.

Al hacerlo, aparecerá nuestra interfaz gráfica, inicialmente con la lista de métodos que han sido implementados y donde podremos seleccionar el método que queremos reproducir.

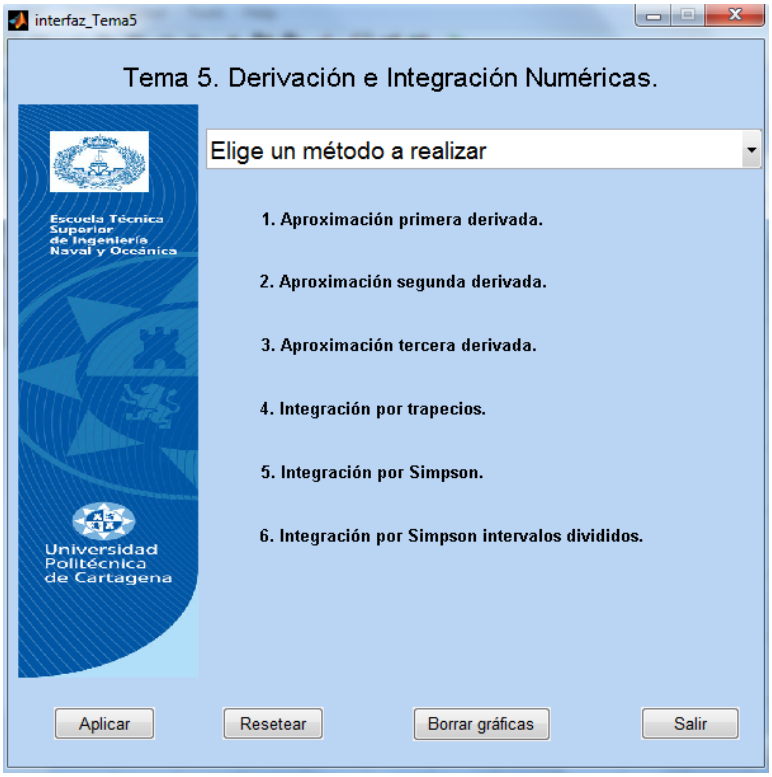


Figura 5.3

En este punto, pulsando en donde dice “Elige un método a realizar” nos aparece automáticamente una lista de los métodos de derivación e integración numérica, donde elegiremos el que deseemos llevar a cabo.

1. Aproximación de la primera derivada.

Seleccionamos el método en la pantalla de inicio. Para comenzar, si presionamos el botón denominado “Aproximación primera derivada” nos aparece en pantalla un pdf con la teoría y ejemplos necesarios para comprender el método.

Probaremos este programa hallando el valor de la primera derivada de la función $f(x) = \cos(x)$, en el punto $c = 1$ y con un paso de discretización $h = 1/10$.

Probamos el programa tanto para tres como cuatro y cinco puntos para así poder observar la diferencia de resultados obtenidos en cada caso y observar el orden de convergencia según el número de puntos y modo de operar.

Para comenzar, debemos introducir los datos en la interfaz gráfica.

Figura 5.4

Comenzamos realizando la aproximación utilizando tan sólo dos puntos. Al pulsar el botón aplicar aparece una ventana emergente con los resultados obtenidos.

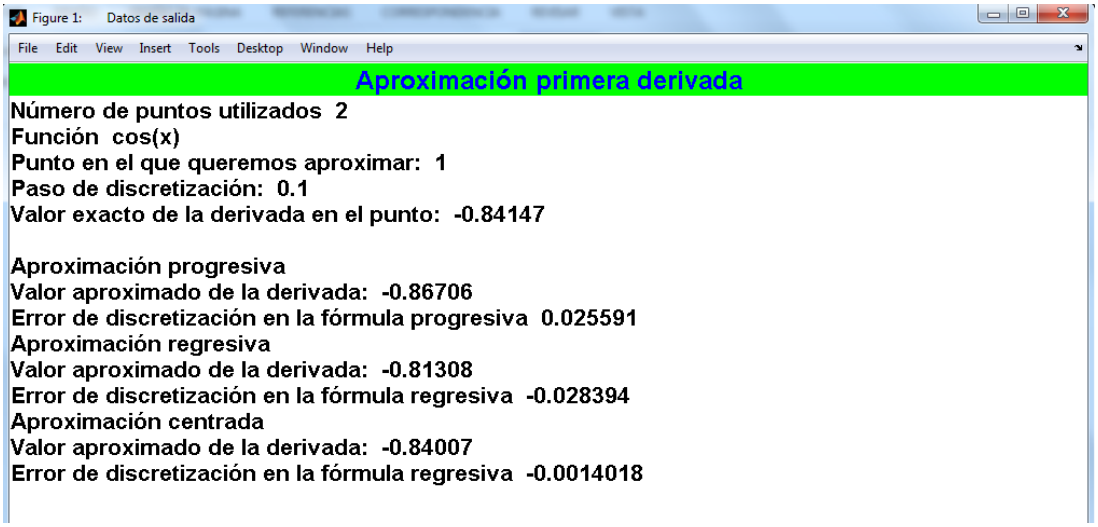


Figura 5.5

Como podemos observar los tres modos de aproximación de la primera derivada mediante dos puntos son bastante precisos, sobresaliendo el último de la forma centrada que, tal y como enunciamos y demostramos en teoría, posee un orden de convergencia mayor, siendo éste de $O(h^2)$, mientras que el de los dos anteriores es $O(h)$.

Probamos ahora a utilizar tres puntos equidistantes. Únicamente cambiamos ese último dato en la interfaz gráfica y al pulsar el botón aplicar aparece una ventana emergente con los resultados obtenidos.

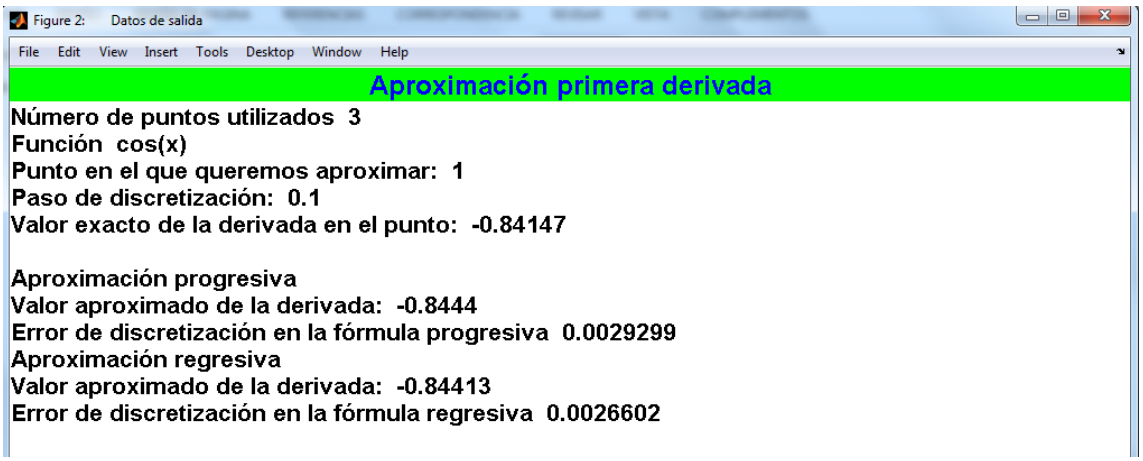


Figura 5.6

Como podemos observar, al añadir un punto en el cálculo de la aproximación hemos aumentado un orden su precisión respecto al caso anterior, siendo ahora su orden de convergencia $O(h^2)$.

Por último probamos a utilizar cuatro puntos equidistantes. Únicamente cambiamos ese último dato en la interfaz gráfica y al pulsar el botón aplicar aparece una ventana emergente con los resultados obtenidos.

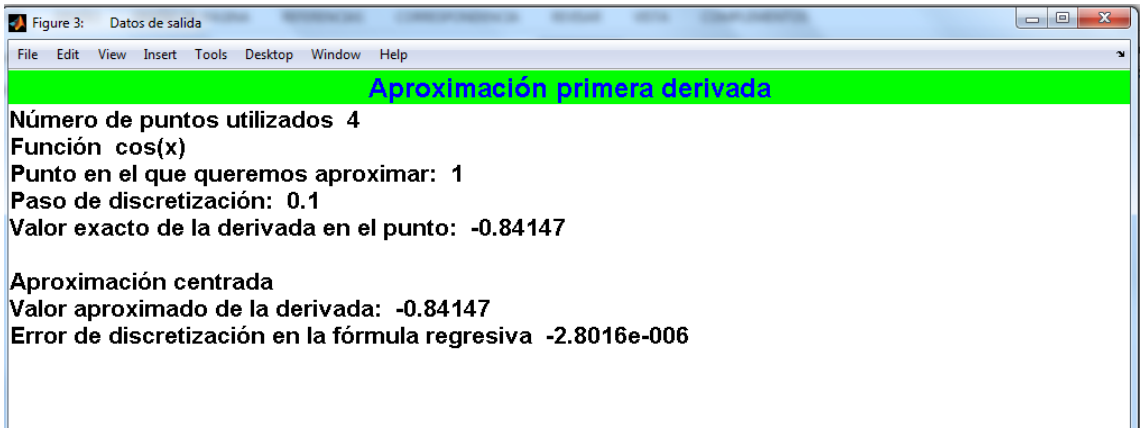


Figura 5.7

Como podemos observar, al añadir un punto más al cálculo de la aproximación ha aumentado en un grado su precisión. Además, al estar el método centrado en torno a un punto, esto aumenta también su precisión en un grado, siendo el orden de convergencia resultante de $O(h^4)$, tal y como enunciamos y demostramos en teoría.

2. Aproximación de la segunda derivada.

Seleccionamos el método en la pantalla de inicio. Para comenzar, si presionamos el botón denominado "Aproximación segunda derivada" nos aparece en pantalla un pdf con la teoría y ejemplos necesarios para comprender el método.

Probaremos este programa aproximando el valor de la segunda derivada de la función

$f(x) = e^{-x^2}$, en el punto $c = 1$ y con un paso de discretización de $h = 1/10$.

Realizaremos la aproximación utilizando en su cálculo tanto tres como cuatro y cinco puntos para así poder observar la diferencia en los resultados obtenidos.

Para ello, primero debemos introducir los datos en la interfaz gráfica del siguiente modo,

Figura 5.8

Para comenzar hemos seleccionado que utilice tres puntos para la aproximación de la segunda derivada en el punto c. Al pulsar el botón aplicar aparece una ventana emergente con los resultados obtenidos,

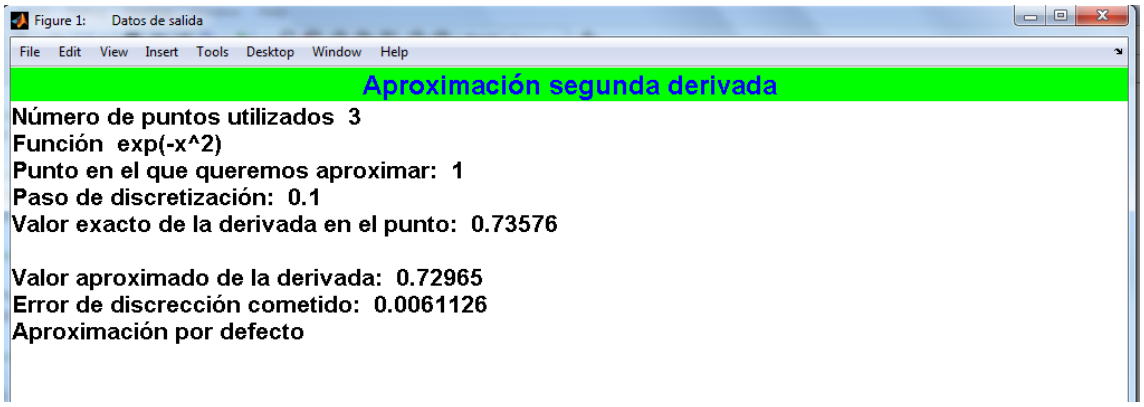


Figura 5.9

Pues bien, como podemos observar este método aproxima con bastante exactitud a la solución exacta, siendo su orden de convergencia de $O(h^2)$, al ser un método centrado en torno al punto c.

Probamos ahora a utilizar cuatro puntos equidistantes. Únicamente cambiamos ese último dato en la interfaz gráfica y al pulsar el botón aplicar aparece una ventana emergente con los resultados obtenidos.

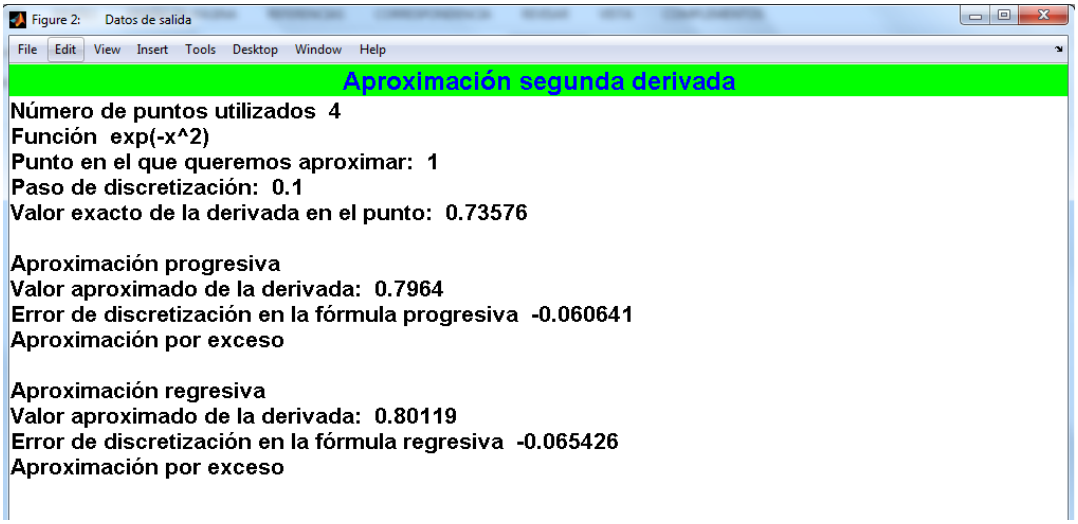


Figura 5.10

Como podemos ver, el orden de precisión en este caso no se ve afectado respecto al caso anterior ya que, a pesar de haber añadido un punto al cálculo de la aproximación, en este caso no se hace de forma centrada, por lo que el orden de convergencia sigue siendo de $O(h^2)$, tal y como vimos en teoría.

Por último probamos a utilizar cinco puntos equidistantes. Únicamente cambiamos ese último dato en la interfaz gráfica y al pulsar el botón aplicar aparece una ventana emergente con los resultados obtenidos.

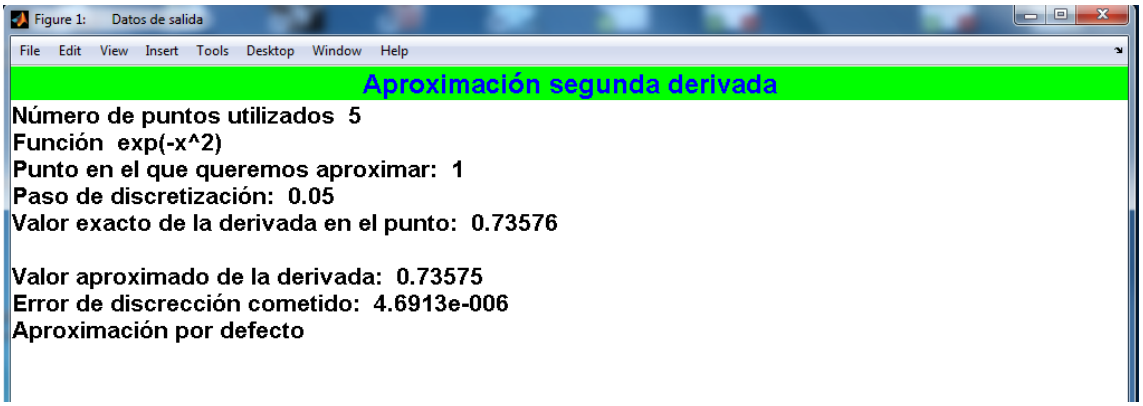


Figura 5.11

Como podemos ver, el orden de precisión se ha visto aumentado notablemente al añadir un punto en el cálculo de la aproximación. Además, al ser el método centrado en torno al punto c esto aumenta también su precisión, siendo su orden de convergencia final de $O(h^4)$, tal y como vimos en teoría.

3. Aproximación de la tercera derivada.

Seleccionamos el método en la pantalla de inicio. Para comenzar, si presionamos el botón denominado “Aproximación tercera derivada” nos aparece en pantalla un pdf con la teoría y ejemplos necesarios para comprender el método.

Probamos este método calculado la aproximación de la tercera derivada de la función

$f(x) = \sqrt{x}$, en el punto $c = 1$ para un paso de discretización $h = \frac{1}{10}$.

En este caso, la aproximación sólo ha sido implementada para utilizar cuatro puntos centrados en torno al punto c, excluyendo a éste.

Para comenzar debemos introducir los datos en la interfaz;

Figura 5.12

Al pulsar el botón aplicar aparece una ventana emergente con los resultados obtenidos,

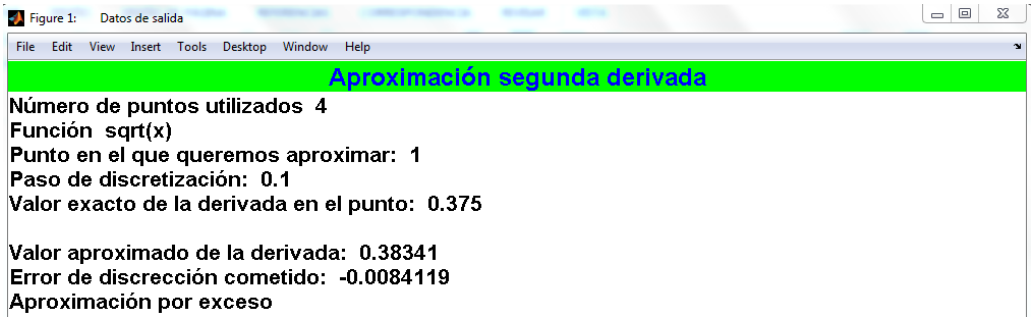


Figura 5.13

Pues bien, como podemos observar este método aproxima con bastante exactitud a la solución exacta, siendo su orden de convergencia de $O(h^2)$, al ser un método centrado en torno al punto c.

Este orden de convergencia podemos observarlo si por ejemplo disminuimos a la mitad el paso de discretización, siendo así $h = 1/20$, y observamos la disminución del error cometido en la aproximación;

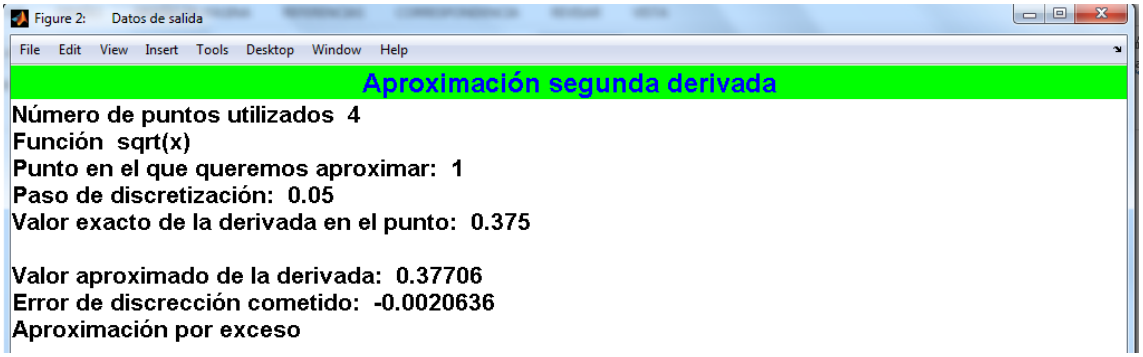


Figura 5.14

4. Integración por trapecios

Seleccionamos el método en la pantalla de inicio. Para comenzar, si presionamos el botón denominado "Integración por trapecios" nos aparece en pantalla un pdf con la teoría y ejemplos necesarios para comprender el método.

Probamos este programa hallando la aproximación de la función cuyos valores en las ordenadas son

F(xi)	0	5.33	6.08	6.251	6.724
xi	0	1.681	3.362	5.043	6.724

Que como vemos está comprendida en el intervalo $[0\ 6{,}724]$ a una distancia equidistante entre puntos de $h = 1.681$.

Para ello debemos introducir los valores en la interfaz gráfica. En primer lugar debemos seleccionar la opción “valores de la función” y pulsar el botón “crear valores función”. Ahí, el programa nos remite a un archivo en el que podremos introducir los valores de los que disponemos de nuestra función. Si, por el contrario, dispusiéramos de la expresión de la función seleccionaríamos la opción “Expresión de la función” y la introduciríamos a continuación.

Además debemos introducir el intervalo en el que queremos realizar la integración y la distancia equidistante entre los puntos que utilizamos para calcularla.

Figura 5.15

Una vez hemos introducido y guardado los datos necesarios, al pulsar el botón aplicar aparece una ventana emergente con los resultados obtenidos;

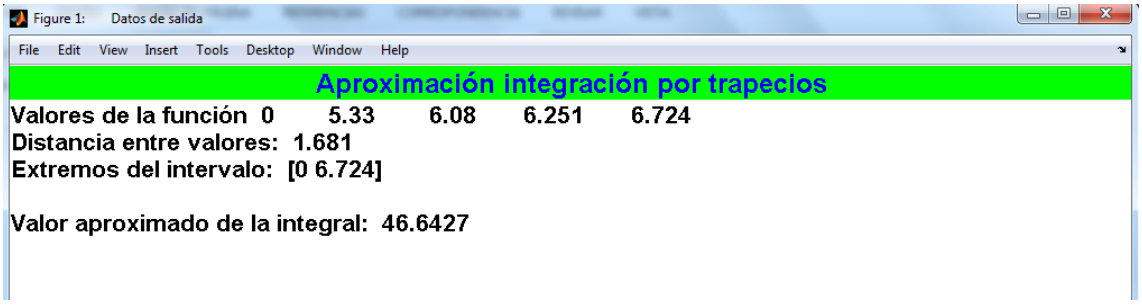


Figura 5.16

A pesar de no poder comprobarlo en este momento, al no disponer de la función explícita de la función, al realizar la aproximación de la integración por métodos más precisos, como haremos a continuación, vemos que el error cometido por este método es significativo. Este modo de aproximación es, sin embargo, útil cuando tenemos tan solo uno o pocos más intervalos de integración.

5. Integración por Simpson.

Seleccionamos el método en la pantalla de inicio. Para comenzar, si presionamos el botón denominado "Integración por Simpson" nos aparece en pantalla un pdf con la teoría y ejemplos necesarios para comprender el método.

Probamos este método con el mismo ejemplo descrito en el caso anterior para así poder apreciar la diferencia en los resultados obtenidos.

Debemos, al igual que en el caso anterior, seleccionar la opción "valores de la función" e introducir estos en el archivo que será abierto al pulsar el botón "crear valores función.

De igual modo, introducimos la distancia entre los puntos equidistantes y los valores de los extremos de nuestro intervalo de integración.

Figura 5.17

Una vez introducidos y guardados los datos necesarios, al pulsar el botón aplicar aparece una ventana emergente con los resultados obtenidos;

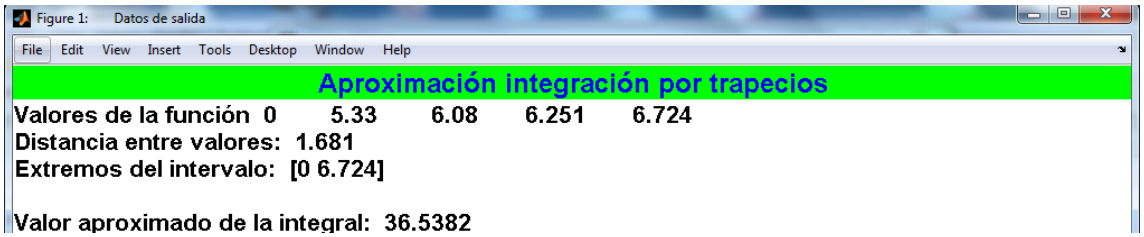


Figura 5.18

Tal y como podemos observar, en este ocasión se realiza una aproximación de la integración mucho más precisa y ajustada que en el caso anterior mediante el método de trapecios.

6. Método de Simpson por intervalos divididos.

Seleccionamos el método en la pantalla de inicio. Para comenzar, si presionamos el botón denominado "Simpson intervalos divididos" nos aparece en pantalla un pdf con la teoría y ejemplos necesarios para comprender el método.

Este programa es en realidad una adaptación del método de integración de Simpson compuesto en el que podemos seleccionar distintos intervalos de integración con una distinta distancia equidistante entre puntos en cada intervalo. Esto resulta ser de gran utilidad en un caso práctico, por ejemplo, de ingeniería naval en el que necesitamos más precisión a la hora de integrar en ciertas zonas del casco, en los casos más habituales propa y popa, mientras que en la zona central del buque podemos utilizar una distancia entre puntos mayor.

Para probar este programa suponemos que tenemos un buque cuya eslora entre perpendiculares o longitud total es $L_{pp} = 100\ m$

Queremos saber el área encerrado por una línea de agua descrita por los siguientes valores discretos;

xi	0	1/2	1	2	3	4	5	6	7	8	9	$9_{1/2}$	10
F(xi)	0	2.4	5.6	9.1	10	10.1	10.1	10.1	9.9	8.8	4.9	2.4	0

Es decir, tenemos tres subintervalos distintos de los cuales en el primero la distancia entre puntos será $h = 5\ m$, que nos sirve para obtener mayor precisión en la zona de popa, en el segundo e intermedio será $h = 10\ m$, y el tercer subintervalo con $h = 5\ m$ que, de igual modo, nos sirve para obtener mayor precisión en la zona de proa.

Para introducir estos valores en la interfaz gráfica primero debemos seleccionar la opción "Valores de la función" y a continuación introducir estos en el archivo al que nos es remitido cuando pulsamos el botón "Crear valores función". Al igual que en los casos anteriores, si dispusiéramos de la expresión de la función no habría más que seleccionar la opción "Expresión de la función" e introducir ésta en la casilla correspondiente.

A continuación serán introducidos los valores de las distintas distancias equidistantes entre puntos en los distintos intervalos en un archivo al que el programa nos remite al pulsar el botón "Equiespaciados de los intervalos". Por último, debemos introducir los puntos en los que comienzan y acaban cada uno de los distintos intervalos en un archivo al que el programa nos remite al pulsar el botón "Puntos de comienzo de cada subintervalo". En dicho archivo se detalla el modo en el que debe ser introducida dicha información con un caso a modo de ejemplo.

Figura 5.19

Una vez introducidos y guardados los datos necesarios, al pulsar el botón aplicar aparece una ventana emergente con los resultados obtenidos;

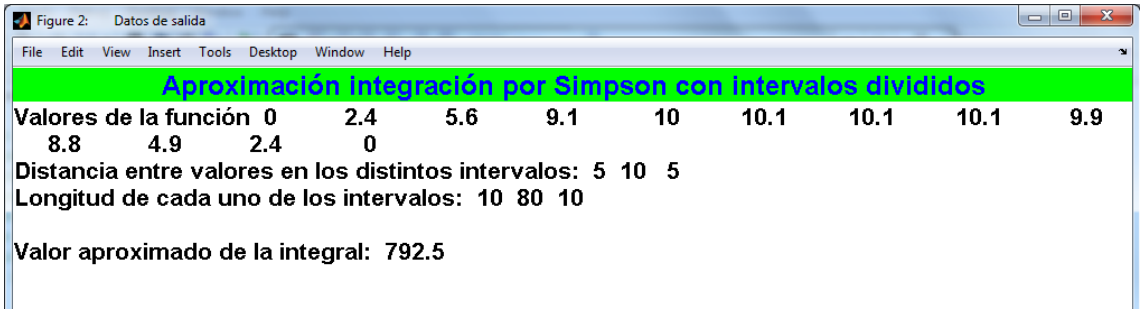


Figura 5.20

De este modo sumos capaces de realizar la integración de una función exigiendo mayor precisión en aquellas zonas en las que sabemos que la función varía más notablemente y necesitamos por tanto un mayor detalle

Capítulo 6. Ecuaciones diferenciales ordinarias.

Introducción.

Las ecuaciones diferenciales se usan a habitualmente para construir modelos matemáticos de problemas de la ciencia y la ingeniería. A menudo se da el caso de que no existe una única solución analítica conocida, por lo que necesitamos aproximaciones numéricas. Los secretos del mundo no se suelen esconder detrás de fórmulas explicitas; en vez de eso, lo que normalmente somos capaces de medir es cómo los cambios de una variable afectan a otra variable. Cuando traducimos esto en un modelo matemático, el resultado es una ecuación diferencial que involucra la velocidad de cambio de una función desconocida y, en la mayoría de los casos, las variables dependiente e independiente.

Consideremos la temperatura $y(t)$ de un objeto que se enfría. Podríamos conjeturar que la velocidad de cambio de la temperatura del cuerpo está relacionada con la diferencia entre su temperatura y la del medio que lo rodea; los experimentos confirman esta conjetura y la ley de enfriamiento de Newton establece que dicha velocidad de cambio es directamente proporcional a la diferencia de temperaturas. Si denotamos por A la temperatura del medio que lo rodea y por $y(t)$ la temperatura del cuerpo en el instante t, entonces

$$\frac{dy}{dt} = -k(y - A) \qquad (1)$$

Donde k es una constante positiva; hace falta incluir el signo negativo ya que la temperatura decrece siempre que la temperatura del cuerpo sea mayor que la del medio.

Si conocemos la temperatura y_0 del cuerpo en el instante $t = 0$, entonces incluimos esta información, que se denomina condición inicial, en el enunciado del problema, de manera que lo que queremos resolver es

$$\frac{dy}{dt} = -k(y - A) \;\; con\; y(0) = y_0. \qquad (2)$$

La solución la calculamos usando la técnica de separación de variables, obteniendo

$$y = A + (y_0 - A)e^{-kt}. \qquad (3)$$

Cada elección de y_0 nos proporciona una solución distinta; es como si el valor inicial fuera el punto de anclaje de la curva correspondiente a la solución.

Problemas de valor inicial.

Una solución del problema de valor inicial

$$y' = f(t, y) \quad con\; y(t_0) = y_0 \qquad (4)$$

En un intervalo $[t_0, t_1]$ es una función derivable $y = y(t)$ tal que

$$y(t_0) = y_0 \quad e \;\; y'(t) = f\big(t, y(t)\big) \;\; para\; todo\; t \in [t_0, t_1]. \qquad (5)$$

Hagamos notar que la gráfica de la solución $y = y(t)$ debe pasar por el punto inicial (t_0, y_0).

En primer lugar, enunciamos el siguiente teorema que garantiza la existencia y unicidad de la solución del problema de valor inicial planteado anteriormente.

Dado el rectángulo $R = \{(t, y):\ a \leq t \leq b,\ c \leq y \leq d\}$, supongamos que $f(t, y)$ es continua en R. Se dice que la función f verifica una **condición de *Lipschitz*** con respecto a su variable y en R si existe una constante $L > 0$ tal que

$$|f(t, y_1) - f(t, y_2)| \leq L|y_1 - y_2| \qquad (6)$$

Para cualesquiera $(t, y_1), (t, y_2) \in R$. La constante L se llama **constante de *Lipschitz*** de *f.*

Damos aquí una condición suficiente para que $f(t, y)$ verifique una condición de *Lipschitz* del tipo anterior.

Si $f(t, y)$ y $\frac{\delta f(t,y)}{\delta y}$ son continuas para todo $(t, y) \in R$, por el teorema del valor medio se tiene

$$f(t, y_1) - f(t, y_2) = \frac{\partial f(t, \hat{y})}{\partial y}(y_1 - y_2)$$

Con $y_1 < \hat{y} < y_2$ entonces si existe $L = sup_{(t,y)\in D}\left|\frac{\partial f(t,y)}{\partial y}\right|$, y además cumple que $L > 0$ tal que

$$\left|f_y(t, y)\right| \leq L \quad para\ todo\ (t, y) \in R$$

Entonces f verifica una condición de *Lipschitz* con respecto a la variable y en R, siendo L su constante de *Lipschitz.*

Expresión general métodos de un paso.

Entre los métodos numéricos más frecuentes para resolver ecuaciones diferenciales se encuentran llamado métodos de discretización, que consisten en encontrar los valores aproximados de la ecuación diferencial en puntos t_k del intervalo $[t_0, t_0 + \alpha]$.

La notación que usaremos será la siguiente; $y(t_k)$ representará el valor exacto de la solución en el punto t_ke $y_k$será el valor aproximado de la solución que obtenemos cuando aplicamos un método numérico.

El esquema general que siguen los métodos que presentaremos, es el siguiente:

1. Se inicia realizando una partición del intervalo $[t_0, t_0 + \alpha]$ en N partes iguales.

   $t_j = t_0 + jh,\ \ j = 0,1, \dots, N, \quad h = \frac{\alpha}{N}$.
2. Conocemos el valor inicial $y(t_0) = y_0$.
3. El resto de $y_k, y_{k+1}, \dots$ se calculan de forma progresiva , en función de los k anteriores

Llamaremos método de un paso a todo algoritmo en el que a partir de los datos t_k, y_k

(aproximación de la solución en el punto t_k del intervalo) y el paso h, podemos calcular el valor y_{k+1} que aproxima a la solución del problema de valor inicial en el punto $t_k + h$.

En este tipo de métodos, la solución se aproxima en un conjunto finitos de puntos que llamaremos nodos. Un método elemental de la forma $y_{k+1} = y_k + h\phi(t_k, y_k)$, para cierta función ϕ llamada función incremental, se dice que es de paso simple, o de un solo paso, porque en el cálculo del nuevo punto sólo interviene de manera explícita el punto inmediatamente anterior.

Cuando usamos un método de variable discreta para resolver de manera aproximada un problema de valor inicial, existen dos fuentes de error: la discretización y el redondeo.

Error de discretización. Supongamos que $\{(t_k, y_k)\}_{k=0}^{M}$ es un conjunto finito de aproximaciones a la solución única $y = y(t)$ de un problema de valor inicial.

El error de truncamiento global o **error de discretización global** e_k se define como

$$e_k = y(t_k) - y_k \quad para\ k = 0,1, \dots, M. \qquad (8)$$

Este error es la diferencia entre la solución exacta y la calculada con el método en el nodo correspondiente.

El error de truncamiento local o **error de discretización local** ε_{k+1} se define como

$$\varepsilon_{k+1} = y(t_{k+1}) - y_k - h\phi(t_k, y_k)\ \ para\ k = 0,1, \dots, M-1. \qquad (9)$$

Este error es el que se comete en un solo paso, el que nos lleva desde el nodo t_k hasta el nodo t_{k+1}

Método de Euler.

No todos los problemas de valor inicial pueden resolverse explícitamente; con frecuencia es imposible hallar una fórmula que presente la solución $y(t)$. Por ejemplo, no existe una "expresión cerrada" para la solución del problema de valor inicial $y' = t^3 + y^2 con\ \ y(0) = 0$. En consecuencia, es necesario disponer de métodos que aproximen la solución de problemas que aparecen en la ciencia y la ingeniería.

El primer método que veremos es el método de Euler y nos servirá para ilustrar una serie de conceptos que juegan un papel importante en los métodos más avanzados. El método de Euler no se utiliza en la práctica debido a que la solución que proporciona acumula errores apreciables a lo largo del proceso; sin embargo, es importante estudiarlo porque es más fácil llevar a cabo el análisis del error de este método que el de otros más exactos pero más complejos.

El valor de y_k lo encontraremos del valor anterior y_{k-1}. Por tanto, estamos ante un método de un sólo paso. Consiste en dividir el intervalo $[t_0, t_0 + \alpha]$ en N partes iguales,

$$t_1 = t_0 + h, \quad t_2 = t_0 + 2h, \dots, \qquad t_N = t_0 + Nh = t_0 + \alpha, \quad h = \frac{\alpha}{M} \qquad (10)$$

Si aplicamos la definición de derivada,

$$y'(t_k) = \lim_{h \to \infty} \frac{y(t_k + h) - y(t_k)}{h} \qquad (11)$$

Deducimos que para h "suficientemente pequeño"

$$y'(t_k) = f\big(t_k, y(t_k)\big) \approx \frac{y(t_k + h) - y(t_k)}{h} \qquad (12)$$

Por tanto,

$$y(t_{k+1}) \approx y(t_k) + hf\big(t_k, y(t_k)\big), \qquad k = 0,1, \dots, M-1. \quad (13)$$

La igualdad anterior nos sugiere el cálculo de los y_k mediante la ley de recurrencia,

$$y_{k+1} = y_k + hf(t_k, y_k) \quad k = 0,1, \dots, M-1. \quad (14)$$

Partiendo de $y(0) = y_0$. La ley (14) se conoce como método de Euler.

Interpretación gráfica.

Si partimos del punto (t_0, y_0), calculamos el valor de la pendiente $m_0 = f(t_0, y_0)$, nos movemos horizontalmente una distancia h y verticalmente una distancia $hf(t_0, y_0)$, entonces lo que hacemos es desplazarnos a lo largo de la recta tangente a la curva $y(t)$ terminando en el punto (t_1, y_1), no perteneciendo este punto a la curva deseada, aunque es la aproximación que se genera. Ahora debemos usar (t_1, y_1), como si fuera un punto correcto para calcular la pendiente $m_1 = f(t_1, y_1)$, y usar este valor para obtener el siguiente desplazamiento vertical $hf(t_1, y_1)$, que nos lleva al punto (t_2, y_2), y así sucesivamente.

Precisión del método de Euler.

Sea $y(t)$ la solución del problema de valor inicial, si $y(t) \in C^2[t_0, b]$ y $\{(t_k, y_k)\}_{k=0}^{M}$ es la sucesión de aproximaciones generada por el método de Euler, entonces

$$|e_k| = |y(t_k) - y_k| = O(h),$$

$$|\varepsilon_{k+1}| = |y(t_{k+1}) - y_k - hf(t_k, y_k)| = O(h^2) \qquad (15)$$

El error final del intervalo se llama error global final y viene dado por

$$E(y(b), h) = |y(b) - y_M| = O(h).$$

Si calculásemos las aproximaciones usando como tamaños de paso h y $h/2$, deberíamos tener

$$E(y(b), h) \approx Ch$$

Para el tamaño de paso más grande y

$$E\left(y(b), \frac{h}{2}\right) \approx C\frac{h}{2} = \frac{1}{2}Ch \approx \frac{1}{2}E(y(b), h). \quad (16)$$

Por lo tanto, si reducimos a la mitad el tamaño de paso en el método de Euler, entonces cabe esperar que el error final se reduce también a la mitad.

Métodos de Taylor de orden superior.

El método de Euler lo hemos deducido de la definición de derivada, pero también puede obtenerse a partir del desarrollo de Taylor de orden $n = 1$ de la función $y(t)$ en el punto t_k. Podemos encontrar un método que mejore la solución del problema de valor inicial, si el desarrollo de Taylor se extiende hasta el orden n.

$$y(t_{k+1}) = y(t_k) + hy'(t_k) + \frac{h^2}{2}y''(t_k) + \cdots + \frac{h^n}{n!}y^{n)}(t_k) + \frac{h^{n+1}}{(n+1)!}y^{n+1)}(\xi_k)$$

$$Con\ \xi_k \in (t_k, t_k + h) \qquad (17)$$

Si la función $f(t,y)$ es "suficientemente regular", entonces podemos calcular las derivadas sucesivas de $y(t)$. En efecto,

$$y'(t) = f\big(t, y(t)\big) = f^{(0)}(t,y)$$

$$y''(t) = \frac{dy'}{dt} = \frac{\partial f}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt} = \frac{\partial f}{dt} + f\frac{\partial f}{\partial y} = f^{(1)}(t,y)$$

$$y'''(t) = \frac{dy''}{dt} = \frac{\partial f^{(1)}}{dt} + \frac{\partial f^{(1)}}{\partial y}\frac{dy}{dt} = \frac{\partial f^{(1)}}{dt} + f\frac{\partial f^{(1)}}{\partial y} = f^{(2)}(t,y)$$

…

$$y^{n)}(t) = \frac{dy^{n-1)}}{dt} = \frac{\partial f^{(n-2)}}{dt} + \frac{\partial f^{(n-2)}}{\partial y}\frac{dy}{dt} = \frac{\partial f^{(n-2)}}{dt} + f\frac{\partial f^{(n-2)}}{\partial y} = f^{(n-1)}(t,y)$$

El método de Taylor de orden n consiste en calcular y_k aplicando la fórmula siguiente;

$$y_{k+1} = y_k + hf^{(0)}(t_k, y_k) + \frac{h^2}{2!}f^{(1)}(t_k, y_k) + \cdots + \frac{h^n}{n!}f^n(t_k, y_k) \qquad (18)$$

Notemos que el método de Euler es un caso particular de Taylor para $n = 1$.

Precisión del método de Taylor de orden n.

El método de Taylor de orden n tiene la propiedad de que el error global final es de orden $O(h^{n+1})$; por tanto, se puede elegir n de manera que este error sea tan pequeño como queramos. Si fijamos el orden n entonces es teóricamente posible fijar el tamaño de paso h de forma que el error global final sea tan pequeños como queramos; sin embargo, en la práctica lo que se hace es construir dos conjuntos de aproximaciones usando tamaños de paso $h\ y\ h/2$ y comparar los resultados.

Supongamos que $y(t)$ es la solución del problema de valor inicial. Si $y(t) \in C^{n+1}[t_0, b]$ y $\{(t_k, y_k)\}_{k=0}^{M}$ es la sucesión de aproximaciones generadas por el método de Taylor de orden n, entonces

$$|e_k| = |y(t_k) - y_k| = O(h^{n+1}),$$

$$|\varepsilon_{k+1}| = |y(t_{k+1}) - y_k - hT_n(t_k, y_k)| = O(h^n). \qquad (19)$$

En particular, el error global en el extremo derecho del intervalo es

$$E(y(b), h) = |y(b) - y_M| = O(h^n).$$

Para el caso de $n = 4$, si calculásemos las aproximaciones tomando como tamaños de paso h y $h/2$, entonces deberíamos tener

$$E(y(b), h) \approx Ch^4$$

Para el tamaño de paso más grande y

$$E\left(y(b), \frac{h}{2}\right) \approx C\frac{h^4}{16} = \frac{1}{16}Ch^4 \approx \frac{1}{16}E(y(b), h). \quad (20).$$

Es decir que si el tamaño de paso se reduce a la mitad, entonces el error global final debería reducirse en un factor de orden 1/16.

El método de Taylor lleva asociada la dificultad de tener que calcular derivadas elevadas de $f(t, y)$, con dificultad nada despreciable, ya que los cálculos pueden hacerse muy lentos. En ocasiones no se conoce la expresión explicita de $f(t, y)$, sino que se dispone de una colección de datos experimentales, por lo que los cálculos con derivadas elevadas tienen grandes posibilidades de ser inexactos. Por esta razón, los métodos multitérminos de Taylor se usan muy raras veces.

Los métodos de Runge-Kutta.

Los métodos de Taylor de la sección precedente tienen la característica deseable de que el error global final es de orden $O(h^n)$, de manera que podemos escoger n tan grande como queramos para que el error sea tan pequeño como deseemos. Sin embargo, los métodos de Taylor presentan dos inconvenientes: la necesidad de determinar n a priori y el cálculo de las derivadas de orden superior, que puede ser bastante complicado. Los métodos de Runge-Kutta se construyen a partir de un método de Taylor, digamos de orden n, de tal manera que el error global final sea del mismo orden $O(h^n)$ pero se evite la evaluación de las derivadas parciales; esto puede conseguirse a cambio de evaluar, en cada caso, la función en varios puntos.

Partimos de la ecuación diferencial con condiciones iniciales

$$\begin{cases} y' = f(t, y) \\ y(t_0) = y_0 \end{cases}$$

En general, un método de Runge-Kutta explícito de m etapas es de la forma

$$y_k = y_{k-1} + \sum_{j=1}^{m} b_j g_j, \qquad (21)$$

Donde

$$\begin{cases} g_1 = hf(t_{k-1}, y_{k-1}) \\ g_2 = hf(t_{k-1} + c_2 h, y_{k-1} + a_{21} g_1), \\ g_3 = hf(t_{k-1} + c_3 h, y_{k-1} + a_{31} g_1 + a_{32} g_2) \\ \dots \\ g_m = hf(t_{k-1} + c_m h, y_{k-1} + a_{m1} g_1 + a_{m2} g_2 + \cdots + a_{m-1} g_{m-1}) \end{cases} \quad (22)$$

Siendo $c_j, j = 2, \dots, m, b_j = 1, \dots, m \; y \; a_{jk}, j = 1, \dots, m, k = 1, \dots, j-1,$ los coeficientes del método. Normalmente estos coeficientes se agrupan según la tabla

$$\begin{array}{c|l} 0 & \\ c_2 & a_{21} \\ c_3 & a_{31}\ a_{32} \\ \dots & \dots\ \dots\dots\ \dots \\ c_m & a_{m1}\ \ a_{m2}\ \dots\ a_{mm-1} \\ \hline & b_1 \quad b_2 \quad \dots \quad b_{m-1} \quad b_m \end{array}$$

Tabla que denominamos por tabla de Butcher, y en su forma matricial

$$\begin{array}{c|c} c^t & A \\ \hline & b \end{array}$$

Donde $c = (0, c_2, \dots, c_m),\ b = (b_1, b_2, .., b_m)$ y $A = \left(a_{jk}\right) \in a_{jk}(R) con\ a_{jk} = 0\ si\ k > j.$ Se satisfacen en general las condiciones de simplificación.

$$c_j = \sum_{k=1}^{j-1} a_{jk} \quad j = 2, .., m.$$

El método de 3 etapas.

Veamos cómo se genera el método de Runge-Kutta de tres etapas que tendrá un error de truncamiento $t_i \approx O(h^4)$. Como sabemos, su tabla de Butcher será,

$$\begin{array}{c|ll} 0 & & \\ c_2 & a_{21} & \\ c_3 & a_{31} & a_{32} \\ \hline & b_1 & b_2 \quad b_3 \end{array}$$

Y el método será de la forma

$$\begin{array}{c} g_1 = hf(t_{k-1}, y_{k-1}) \\ g_2 = hf(t_{k-1} + c_2 h, y_{k-1} + a_{21} g_1) \\ g_3 = hf(t_{k-1} + c_3 h, y_{k-1} + a_{31} g_1 + a_{32} g_2) \end{array} \quad (23)$$

Y

$$y_k = y_{k-1} + b_1 g_1 + b_2 g_2 + b_3 g_3 \qquad (24)$$

Por otra parte, la aproximación mediante la serie de Taylor de orden tres de $y(t; t_{k-1}, y_{k-1})$ es

$$y_k = y_{k-1} + f(t_{k-1}, y_{k-1})h + \frac{1}{2}\left(\frac{\partial}{\partial t} f(t_{k-1}, y_{k-1}) + f(t_{k-1}, y_{k-1})\frac{\partial}{\partial y} f(t_{k-1}, y_{k-1})\right)h^2 \quad (25)$$

$$+\frac{1}{6}\left(\frac{\partial^2}{\partial t^2} f(t_{k-1}, y_{k-1}) + 2f(t_{k-1}, y_{k-1})\frac{\partial^2}{\partial y \partial t} f(t_{k-1}, y_{k-1})\right.$$
$$+ \frac{\partial}{\partial t} f(t_{k-1}, y_{k-1})\frac{\partial}{\partial y} f(t_{k-1}, y_{k-1})$$
$$\left. + f(t_{k-1}, y_{k-1})\left(\frac{\partial}{\partial y} ft(t_{k-1}, y_{k-1})\right)^2 f(t_{k-1}, y_{k-1})^2 \frac{\partial^2}{\partial y^2} f(t_{k-1}, y_{k-1})\right)h^3$$
$$+ O(h^4) \qquad (26)$$

Tomamos la función

$$G_2(h) = f(t_{k-1} + c_2 h, y_{k-1} + a_{21} g_1)$$

Derivamos dos veces respecto de h

$$G'_2(h) = c_2 \frac{\partial}{\partial t} f(t_{k-1} + c_2 h, y_{k-1} + a_{21} g_1) + \frac{\partial}{\partial y} f(t_{k-1} + c_2 h, y_{k-1}$$
$$+ a_{21} g_1) a_{21} f(t_{k-1}, y_{k-1})$$

$$G''_2(h) = c_2^2 \frac{\partial^2}{\partial t^2} f(t_{k-1} + c_2 h, y_{k-1} + a_{21} g_1)$$
$$+ 2c_2 \frac{\partial^2}{\partial t \partial y} f(t_{k-1} + c_2 h, y_{k-1} + a_{21} g_1) a_{21} f(t_{k-1}, y_{k-1})$$
$$+ \frac{\partial^2}{\partial y^2} f(t_{k-1} + c_2 h, y_{k-1} + a_{21} g_1) a_{21}^2 f(t_{k-1}, y_{k-1})^2,$$

De donde el desarrollo de Taylor de orden 2 es

$$G_2(h) = G_2(0) + G'_2(0)h + \frac{1}{2} G''_2(0)h^2 + O(h^3)$$

$$= f(t_{k-1}, y_{k-1}) + \left(c_2 \frac{\partial}{\partial t} f(t_{k-1}, y_{k-1}) + a_{21} f(t_{k-1}, y_{k-1})\frac{\partial}{\partial y} f(t_{k-1}, y_{k-1})\right)h$$

$$+\frac{1}{2}\left(c_2^2 \frac{\partial^2}{\partial t^2} f(t_{k-1}, y_{k-1}) + 2c_2 a_{21} f(t_{k-1}, y_{k-1})\frac{\partial^2}{\partial t \partial y} f(t_{k-1}, y_{k-1})\right.$$
$$\left. + a_{21}^2 f(t_{k-1}, y_{k-1})^2 \frac{\partial^2}{\partial y^2} f(t_{k-1}, y_{k-1})\right)h^2 + O(h^3)$$

Y por otra parte

$$G_3(h) = f\big(t_{k-1} + c_3 h, y_{k-1} + a_{31} g_1 + a_{32} h G_2(h)\big)$$

Que derivando dos veces nos da

$$G'_3(h) = c_3 \frac{\partial}{\partial t} f\big(t_{k-1} + c_3 h, y_{k-1} + a_{31} g_1 + a_{32} h G_2(h)\big) + \frac{\partial}{\partial y} f\big(t_{k-1} + c_3 h, y_{k-1} + a_{31} g_1 + a_{32} h G_2(h)\big) \cdot \Big(a_{31} f(t_{k-1}, y_{k-1}) + a_{32}\big(G_2(h) + h G'_2(h)\big)\Big)$$

$$G''_3(h) = c_3^2 \frac{\partial^2}{\partial t^2} f\big(t_{k-1} + c_3 h, y_{k-1} + a_{31} g_1 + a_{32} h G_2(h)\big) +$$

$$+2c_3 \frac{\partial^2}{\partial t \partial y} f\big(t_{k-1} + c_3 h, y_{k-1} + a_{31} g_1 + a_{32} h G_2(h)\big) \cdot \Big(a_{31} f(t_{k-1}, y_{k-1}) + a_{32}\big(G_2(h) + h G'_2(h)\big)\Big) +$$

$$+\frac{\partial^2}{\partial y^2} f\big(t_{k-1} + c_3 h, y_{k-1} + a_{31} g_1 + a_{32} h G_2(h)\big) \cdot \Big(a_{31} f(t_{k-1}, y_{k-1}) + a_{32}\big(G_2(h) + h G'_2(h)\big)\Big)^2 +$$

$$+\frac{\partial}{\partial y} f\big(t_{k-1} + c_3 h, y_{k-1} + a_{31} g_1 + a_{32} h G_2(h)\big) \Big(a_{32}\big(2G'_2(h) + h G''_2(h)\big)\Big)$$

De donde

$$G'_3(0) = c_3 \frac{\partial}{\partial t} f(t_{k-1}, y_{k-1}) + (a_{31} + a_{32}) \frac{\partial}{\partial y} f(t_{k-1}, y_{k-1}) f(t_{k-1}, y_{k-1}),$$

$$G''_3(0) = c_3^2 \frac{\partial^2}{\partial t^2} f(t_{k-1}, y_{k-1}) + 2c_3 (a_{31}, a_{32}) \frac{\partial^2}{\partial t \partial y} f(t_{k-1}, y_{k-1}) f(t_{k-1}, y_{k-1})$$

$$+(a_{31} + a_{32})^2 \frac{\partial^2}{\partial y^2} f(t_{k-1}, y_{k-1}) f(t_{k-1}, y_{k-1})^2$$

$$+2a_{32} \frac{\partial}{\partial y} f(t_{k-1}, y_{k-1}) \left(c_2 \frac{\partial}{\partial t} f(t_{k-1}, y_{k-1}) + a_{21} f(t_{k-1}, y_{k-1}) \frac{\partial}{\partial y} f(t_{k-1}, y_{k-1})\right)$$

Por lo que el desarrollo de Taylor en cero es

$$G_3(h) = G_3(0) + G'_3(0) h + \frac{1}{2} G''_3(0) h^2 + O(h^3)$$

$$= f(t_{k-1}, y_{k-1}) + \left(c_3 \frac{\partial}{\partial t} f(t_{k-1}, y_{k-1}) + (a_{31} + a_{32}) \frac{\partial}{\partial y} f(t_{k-1}, y_{k-1}) f(t_{k-1}, y_{k-1})\right) h$$

$$+\frac{1}{2}\Bigg(c_3^2 \frac{\partial^2}{\partial t^2} f(t_{k-1}, y_{k-1}) + 2c_3 (a_{31} + a_{32}) \frac{\partial^2}{\partial t \partial y} f(t_{k-1}, y_{k-1}) f(t_{k-1}, y_{k-1}) + (a_{31} + a_{32})^2 \frac{\partial^2}{\partial y^2} f(t_{k-1}, y_{k-1}) f(t_{k-1}, y_{k-1})^2 + 2a_{32} \frac{\partial}{\partial y} f(t_{k-1}, y_{k-1}) \cdot \left(c_2 \frac{\partial}{\partial y} f(t_{k-1}, y_{k-1}) + a_{21} f(t_{k-1}, y_{k-1}) \frac{\partial}{\partial y} f(t_{k-1}, y_{k-1})\right)\Bigg) h^2$$

Introducimos los desarrollos de Taylor G_2 y G_3 en la igualdad (24), y comparando con el desarrollo de Taylor de orden 2 (25), obtenemos las ecuaciones

$$\begin{cases} b_1 + b_2 + b_3 = 1 \\ b_2 c_2 + b_3 c_3 = \frac{1}{2} \\ b_2 a_{21} + b_3 (a_{31} + a_{32}) = \frac{1}{2} \\ b_2 c_2^2 + b_3 c_3^2 = \frac{1}{3} \\ b_2 c_2 a_{21} + b_3 c_3 (a_{31} + a_{32}) = \frac{1}{3} \\ b_2 a_{21}^2 + b_3 (a_{31} + a_{32})^2 = \frac{1}{3} \\ b_3 a_{32} c_2 = \frac{1}{6} \\ b_3 a_{32} a_{21} = \frac{1}{6} \end{cases}$$

De las dos últimas obtenemos que $c_2 = a_{21}$con lo que usando la segunda y la tercera, llegamos a $c_3 = a_{31} + a_{32}$. Entonces la quinta y la sexta ecuaciones se simplifican a

$$b_2 c_2^2 + b_3 c_3^2 = \frac{1}{3}$$

Que es la cuarta, y la última ecuación es la antepenúltima, con lo que el sistema reducido de ecuaciones nos queda

$$\begin{cases} b_1 + b_2 + b_3 = 1 \\ c_2 = a_{21} \\ c_3 = a_{31} + a_{32} \\ b_2 c_2 + b_3 c_3 = \frac{1}{2} \\ b_2 c_2^2 + b_3 c_3^2 = \frac{1}{3} \\ b_3 a_{32} c_2 = \frac{1}{6} \end{cases}$$

Que nos dan los métodos de Runge-Kutta de orden tres, que es una familia biparamétrica de métodos numéricos. Si tomamos c_2 y c_3 como parámetros, tenemos las ecuaciones

$$\begin{cases} b_2 c_2 + b_3 c_3 = \frac{1}{2} \\ b_2 c_2^2 + b_3 c_3^2 = \frac{1}{3} \end{cases}$$

Dando lugar a

$$b_2 = \frac{c_3 (\frac{c_3}{2} - \frac{1}{3})}{c_2 c_3 (c_3 - c_2)}$$

$$b_3 = \frac{c_2 (\frac{1}{3} - \frac{c_2}{2})}{c_2 c_3 (c_3 - c_2)}$$

Y de la última

$$a_{32} = \frac{1}{6b_3c_2} = \frac{c_2c_3(c_3 - c_2)}{6(\frac{1}{3} - \frac{c_2}{2})}$$

De $c_3 = a_{31} + a_{32}$ tenemos que

$$a_{31} = c_3 - a_{32} = c_3 - \frac{c_2c_3(c_3 - c_2)}{6(\frac{1}{3} - \frac{c_2}{2})}$$

Y de $b_1 + b_2 + b_3 = 1$ concluimos

$$b_1 = 1 - b_2 - b_3 = 1 - \frac{c_3\left(\frac{c_3}{2} - \frac{1}{3}\right)}{c_2c_3(c_3 - c_2)} - \frac{c_2\left(\frac{1}{3} - \frac{c_2}{2}\right)}{c_2c_3(c_3 - c_2)} = \frac{\left(c_2c_3 + \frac{1}{3}\right)(c_3 - c_2) - \frac{1}{2}(c_3^2 - c_3^2)}{c_2c_3(c_3 - c_2)}$$

Así obtenemos los métodos de Runge-Kutta de orden 3,

0			
½	½		
¾	0	¾	
	$\frac{2}{9}$	$\frac{1}{3}$	$\frac{4}{9}$

Y

0			
½	½		
1	-1	2	
	$\frac{1}{6}$	$\frac{2}{3}$	$\frac{1}{6}$

Estas soluciones son válidas siempre que c_2 y c_3 sean no nulos y distintos. Existen métodos que se obtienen cuando alguna de estas cantidades son nulas, y que se obtienen de igual manera.

Como vemos, dado que el error local de los métodos de Runge-Kutta de tres etapas es de orden 4 $[O(h^4)]$, tenemos que el error global de los métodos de tres etapas es de orden 3.

El método de cuatro etapas.

Procediendo como en el caso anterior de tres etapas, aumentando en un orden los desarrollos de Taylor de las funciones implicadas, tenemos que si la tabla de Butcher de un método de cuatro etapas es

$$\begin{array}{c|cccc} 0 & & & & \\ c_2 & a_{21} & & & \\ c_3 & a_{31} & a_{32} & & \\ c_4 & a_{41} & a_{42} & a_{43} & \\ \hline & b_1 & b_2 & b_3 & b_4 \end{array}$$

Entonces han de satisfacerse la simplificación

$$c_j = \sum_{k=1}^{j-1} a_{jk}, \;\; j = 2,3,4$$

Y las condiciones

$$\begin{cases} b_1 + b_2 + b_3 + b_4 = 1 \\ b_2c_2 + b_3c_3 + b_4c_4 = \frac{1}{2} \\ b_2c_2^2 + b_3c_3^2 + b_4c_4^2 = \frac{1}{3} \\ b_3a_{32}c_2 + b_4(a_{42}c_2 + a_{43}c_3) = \frac{1}{6} \\ b_2c_2^3 + b_3c_3^3 + b_4c_4^3 = \frac{1}{4} \\ b_3c_3a_{32}c_2 + b_4c_4(a_{42}c_2 + a_{43}c_3) = \frac{1}{8} \\ b_3a_{32}c_2^2 + b_4(a_{42}c_2^2 + a_{43}c_3^2) = \frac{1}{12} \\ b_4a_{43}a_{32}c_2 = \frac{1}{24} \end{cases}$$

La primera de las cuales viene de la condición de consistencia de los métodos de Runge-Kutta en general. Butcher en 1963 dio la simplificación

$$\sum_{i=1}^{4} b_i a_{ij} = b_j\big(1 - c_j\big), \;\; j = 2,3,4 \qquad (27)$$

Donde $a_{ij} = 0$ si $i \geq j$, de la cual se tiene para $j = 4$ que

$$b_4(1 - c_4) = 0,$$

De donde $c_4 = 1$ ya que $b_4 \neq 0$. Como las ecuaciones 2,3 y 5 son lineales en $b_2, b_3\ y\ b_4$, tenemos que

$$b_2 = \frac{1 - 2c_3}{12c_2(1 - c_2)(c_2 - c_3)}$$

$$b_3 = \frac{1 - 2c_2}{12c_3(1 - c_3)(c_3 - c_2)}$$

Y

$$b_4 = \frac{6c_2c_3 - 4(c_3 + c_2) + 3}{12c_2(1 - c_2)(1 - c_3)}$$

Tomando ahora $j = 3$ en (27) obtenemos

$$a_{43} = \frac{(1 - c_2)(2c_2 - 1)(1 - c_3)}{c_3(c_2 - c_3)(6c_2c_3 - 4(c_2 + c_3) + 3)}$$

Finalmente, de la última ecuación y (27) con $j = 2$ obtenemos

$$a_{32} = \frac{c_3(c_2 - c_3)}{2c_2(2c_2 - 1)}$$

Y

$$a_{42} = \frac{(1 - c_2)[2(1 - c_3)(1 - 2c_3) - (c_2 - c_3)]}{2c_2(c_2 - c_3)[6c_2c_3 - 4(c_2 + c_3) + 3]}$$

Como vemos, estas soluciones dependientes de dos parámetros son válidas siempre que $c_2 \notin \left\{0, \frac{1}{2}, 1\right\}$, $c_3 \notin \{0,1\}$ y $c_2 \neq c_3$. Cuando algunas de estas condiciones no se satisfacen, existen no obstante métodos de Runge-Kutta con estos coeficientes.

Estos métodos son de orden 4 (orden 5 tiene el error local de truncamiento) y representan una familia de cinco parámetros de métodos cuyos ejemplos son

0				
1/3	1/3			
2/3	-1/3	1		
1	1	−1	1	
	$\frac{1}{8}$	$\frac{3}{8}$	$\frac{3}{8}$	$\frac{1}{8}$

Y sobre todo que cumple $c_2 = c_3$,

0				
½	½			
½	0	½		
1	0	0	1	
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

Que es el método debido a Kutta de 1905 y que usualmente se conoce como el método de Runge-Kutta.

Así podemos obtener los sucesivos puntos sustituyendo cada una de las estimaciones en

$$y_{k+1} = y_k + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \qquad (28)$$

Donde

$$k_1 = f(t_k, y_k)$$
$$k_2 = f\left(t_k + \frac{h}{2}, y_k + k_1\frac{h}{2}\right)$$
$$k_3 = f\left(t_k + \frac{h}{2}, y_k + k_2\frac{h}{2}\right)$$
$$k_4 = f(t_k + h, y_k + k_3 h)$$

Este método es capaz de conseguir precisiones altas sin tener que tomar el paso h tan pequeño como para hacer excesiva la tarea de cálculo.

Sistemas de ecuaciones diferenciales.

En los apartados anteriores hemos estudiado métodos numéricos referidos a problemas de valores iniciales de primer orden. Muchos problemas importantes de la vida cotidiana se modelan utilizando sistemas de ecuaciones diferenciales o bien usando ecuaciones diferenciales de segundo orden u orden superior. Dichas ecuaciones pueden reducirse a un sistema de ecuaciones diferenciales. Por ejemplo, haciendo $x = y'$, la ecuación

$$\frac{d^2y}{dt^2} = f\left(t, y, \frac{dy}{dt}\right)$$

Podemos transformarla en un sistema de ecuaciones diferenciales de primer orden

$$y' = x, \quad x' = f(t, x, y).$$

Generalmente se suelen reducir todas las ecuaciones diferenciales de orden dos a sistemas de dos ecuaciones de primer orden, no obstante, estos resultados pueden generalizarse al caso de más de dos ecuaciones.

Es natural, por tanto, extender los métodos encontrados en la sección anterior a estas nuevas situaciones. Nuestro objetivo será el de formular métodos para aproximar las soluciones de sistemas de ecuaciones del tipo:

$$\begin{cases} x' = f(t,x,y) \\ y' = g(t,x,y) \end{cases} \qquad (29)$$

Con las condiciones iniciales $x(t_0) = x_0$ e $y(t_0) = y_0$.

Para no tener problemas con las soluciones, supondremos que las funciones f y g son "suficientemente regulares" para que exista una única solución al problema de valores iniciales planteado.

Al igual que en los apartados anteriores, deseamos encontrar las soluciones aproximadas de $x(t)$ e $y(t)$ en los puntos

$$t = t_0, \quad t = t_1 = t_0 + h, \dots, t_N = t_0 + Nh.$$

Generalización de los métodos.

Los métodos numéricos estudiados para aproximar las soluciones de ecuaciones diferenciales pueden extenderse sin dificultad para resolver el problema (29). A continuación introduciremos los métodos de *Euler* y Runge-Kutta.

La idea que está detrás del método de Euler es el concepto de la derivada.

$$\frac{dx(t)}{dt} = \lim_{h\to 0} \frac{x(t+h) - x(t)}{h}.$$

Si tomamos como valor de h una cantidad "suficientemente pequeña", entonces nos aparece el sistema dinámico discreto

$$\frac{x(t+h) - x(t)}{h} = f\big(t, x(t), y(t)\big).$$

Ahora podemos definir $x(t_k + h) = x_{k+1}$, $x(t_k) = x_k$, con lo cual

$$x_{k+1} = x_k + hf(t_k, x_k, y_k).$$

Actuando del mismo modo con $g(t,x,y)$, obtenemos

$$\begin{aligned} x_{k+1} &= x_k + hf(t_k, x_k, y_k) \\ y_{k+1} &= y_k + hg(t_k, x_k, y_k) \end{aligned} \qquad (30)$$

De esta manera, conociendo los valores iniciales x_0, y_0, el tamaño de paso h, y las funciones f y g, podemos calcular la sucesión de valores x_k e y_k para $k = 0,1,2, \dots$

En consecuencia, disponemos de un método para encontrar aproximaciones numéricas de las soluciones del sistema (29). Esta técnica se conoce con el nombre de método de *Euler* para sistemas de dos ecuaciones diferenciales.

Para conseguir un grado de precisión razonable, es necesario utilizar un método de orden mayor. Por ejemplo, las fórmulas para el método de Runge-Kutta de orden 4 son

$$x_{k+1} = x_k + \frac{h}{6}(f_1 + 2f_2 + 2f_3 + f_4)$$
$$y_{k+1} = y_k + \frac{h}{6}(g_1 + 2g_2 + 2g_3 + g_4) \qquad (31)$$

Siendo

$f_1 = f(t_k, x_k, y_k)$, $\quad g_1 = g(t_k, x_k, y_k)$

$f_2 = f\left(t_k + \frac{h}{2}, x_k + \frac{h}{2}f_1, y_k + \frac{h}{2}g_1\right)$ $\quad g_2 = g\left(t_k + \frac{h}{2}, x_k + \frac{h}{2}f_1, y_k + \frac{h}{2}g_1\right)$

$f_3 = f\left(t_k + \frac{h}{2}, x_k + \frac{h}{2}f_2, y_k + \frac{h}{2}g_2\right)$ $\quad g_3 = g\left(t_k + \frac{h}{2}, x_k + \frac{h}{2}f_2, y_k + \frac{h}{2}g_2\right)$

$f_4 = f(t_k + h, x_k + hf_3, y_k + hg_3)$ $\quad g_4 = g(t_k + h, x_k + hf_3, y_k + hg_3)$

Ejemplo de aplicación al ámbito naval.

Suponemos que tenemos en nuestro buque un tanque de lastre que contiene 100 litros de agua en los que se han disuelto 20 kg de sal. Por otro lado, otro tanque de lastre contiene 1 kilogramo de sal por litro que es bombeada el primer tanque a razón de 7 litros por minuto. La mezcla resultante es enviada al exterior a razón de 8 litros por minuto. Determinar la función que da la cantidad de sal en cada instante y en qué tiempo se vaciará completamente el tanque.

El enunciado del problema proporciona los siguientes datos:

$A = 20kg, \quad a = 1Kg/L$

$V_0 = 100L, \;\; v_1 = 7l/min \quad v_2 = 8l/min$

La ecuación diferencial que mide la cantidad de sal en cada instante, viene dada por

$$y'(t) + \frac{8}{100 + (7-8)t} y(t) = 7 * 1$$

$$y'(t) + \frac{8}{100 - t} y(t) = 7$$

Tenemos, además la condición inicial de que para el instante $t = 0$, $y(0) = A = 20$.

Resolvemos el problema aplicando, por ejemplo, el método de Euler explícito. Utilizamos una diferencia de tiempo en cada iteración relativamente pequeña ya que conforme aumentamos el tiempo entre cada iteración aumentamos los errores acumulados en el proceso, ya que, tal y como explicamos en teoría, el método de Euler se basa en la definición de derivada.

Introducimos los datos descritos en la interfaz gráfica;

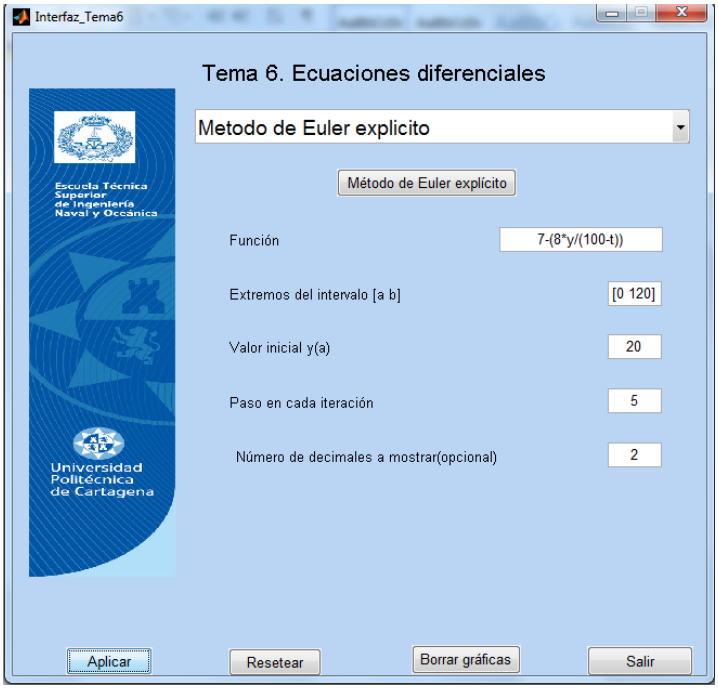
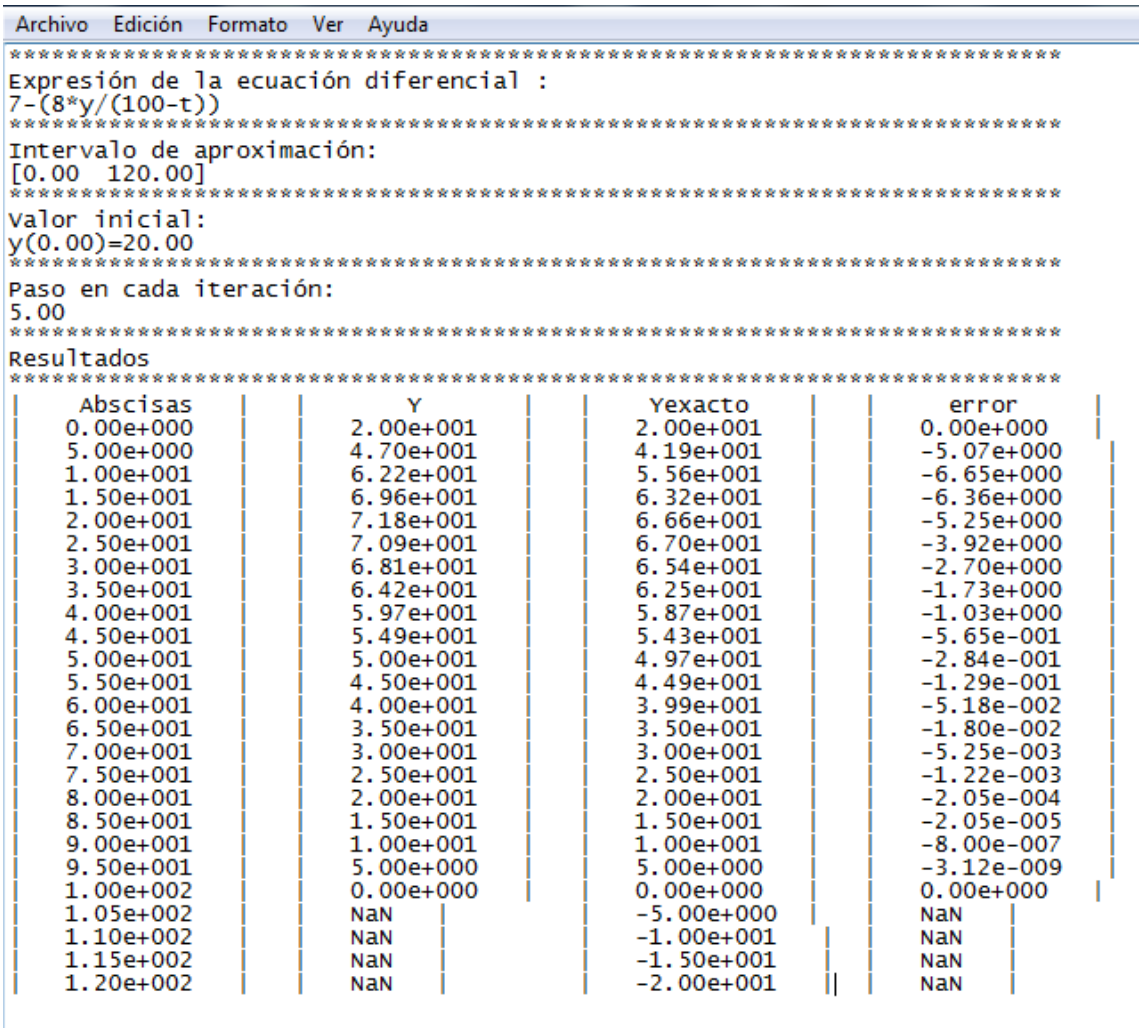


Figura 6.0

Pulsando ahora el botón aplicar obtenemos los siguientes resultados en un archivo de texto;



Archivo Edición Formato Ver Ayuda

```
**********************************************************************************
Expresión de la ecuación diferencial :
7-(8*y/(100-t))
**********************************************************************************
Intervalo de aproximación:
[0.00  120.00]
**********************************************************************************
Valor inicial:
y(0.00)=20.00
**********************************************************************************
Paso en cada iteración:
5.00
**********************************************************************************
Resultados
**********************************************************************************
|     Abscisas     |     |        Y         |     |     Yexacto      |     |      error       |
|    0.00e+000     |     |     2.00e+001     |     |     2.00e+001     |     |     0.00e+000     |
|    5.00e+000     |     |     4.70e+001     |     |     4.19e+001     |     |     -5.07e+000     |
|    1.00e+001     |     |     6.22e+001     |     |     5.56e+001     |     |     -6.65e+000     |
|    1.50e+001     |     |     6.96e+001     |     |     6.32e+001     |     |     -6.36e+000     |
|    2.00e+001     |     |     7.18e+001     |     |     6.66e+001     |     |     -5.25e+000     |
|    2.50e+001     |     |     7.09e+001     |     |     6.70e+001     |     |     -3.92e+000     |
|    3.00e+001     |     |     6.81e+001     |     |     6.54e+001     |     |     -2.70e+000     |
|    3.50e+001     |     |     6.42e+001     |     |     6.25e+001     |     |     -1.73e+000     |
|    4.00e+001     |     |     5.97e+001     |     |     5.87e+001     |     |     -1.03e+000     |
|    4.50e+001     |     |     5.49e+001     |     |     5.43e+001     |     |     -5.65e-001     |
|    5.00e+001     |     |     5.00e+001     |     |     4.97e+001     |     |     -2.84e-001     |
|    5.50e+001     |     |     4.50e+001     |     |     4.49e+001     |     |     -1.29e-001     |
|    6.00e+001     |     |     4.00e+001     |     |     3.99e+001     |     |     -5.18e-002     |
|    6.50e+001     |     |     3.50e+001     |     |     3.50e+001     |     |     -1.80e-002     |
|    7.00e+001     |     |     3.00e+001     |     |     3.00e+001     |     |     -5.25e-003     |
|    7.50e+001     |     |     2.50e+001     |     |     2.50e+001     |     |     -1.22e-003     |
|    8.00e+001     |     |     2.00e+001     |     |     2.00e+001     |     |     -2.05e-004     |
|    8.50e+001     |     |     1.50e+001     |     |     1.50e+001     |     |     -2.05e-005     |
|    9.00e+001     |     |     1.00e+001     |     |     1.00e+001     |     |     -8.00e-007     |
|    9.50e+001     |     |     5.00e+000     |     |     5.00e+000     |     |     -3.12e-009     |
|    1.00e+002     |     |     0.00e+000     |     |     0.00e+000     |     |     0.00e+000     |
|    1.05e+002     |     |     NaN     |     |     -5.00e+000     |     |     NaN     |
|    1.10e+002     |     |     NaN     |     |     -1.00e+001     |     |     NaN     |
|    1.15e+002     |     |     NaN     |     |     -1.50e+001     |     |     NaN     |
|    1.20e+002     |     |     NaN     |     |     -2.00e+001     |     |     NaN     |
```

Figura 6.1

Además de una gráfica en la que es representado tanto la función explícita de la expresión que define la solución de la ecuación diferencial en el tiempo de manera exacta, como las aproximaciones obtenidas por el método de Euler.

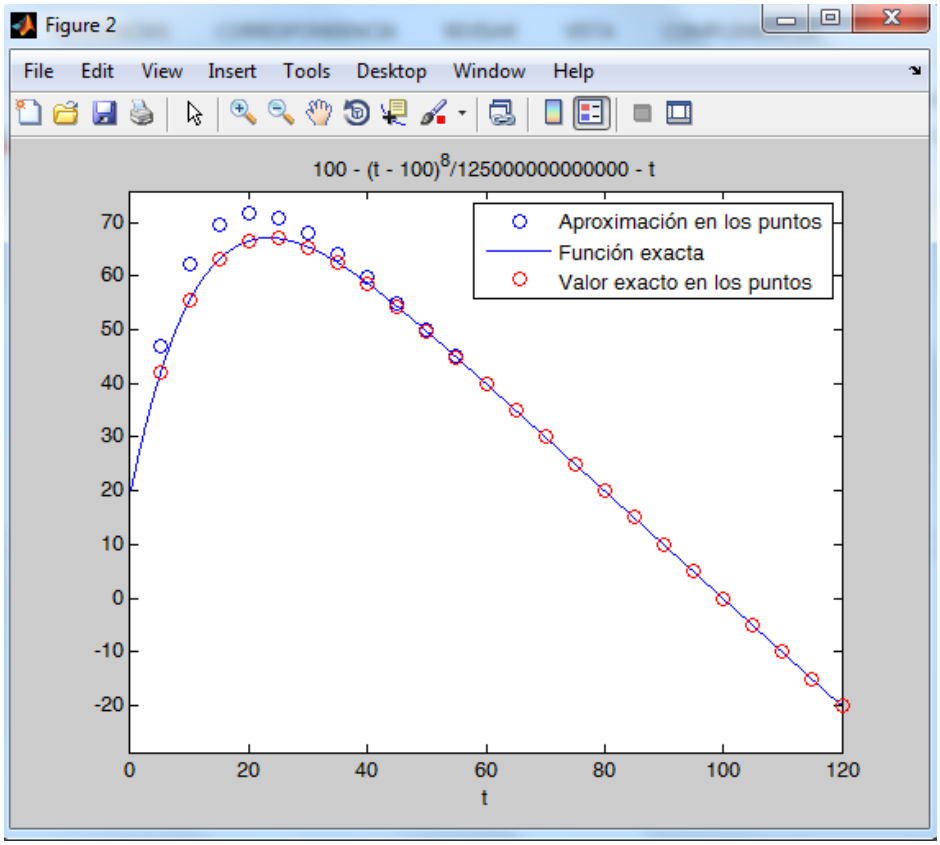


Figura 6.2

Como podemos observar, tanto en la representación de la función explicita exacta en la gráfica, como en los valores obtenidos de esta, la función explícita obtenida se iguala a cero cuando $t = 100\ min$, es decir, el tanque quedará vacío una vez transcurrido ese tiempo.

Sin embargo, como también podemos observar, las aproximaciones realizadas se acercan al resultado real conforme se avanza en el tiempo hasta que alcanza los 100 min. Esto se debe a no está definida la aproximación para valores negativos.

Hoy en día se continúa avanzando en el estudio de métodos más sofisticados de aproximación de resultados de las ecuaciones diferenciales, con el fin de predecir con mayor exactitud dichos resultados, en campos de estudio como, por ejemplo, el meteorológico.

Ayuda a la interfaz gráfica.

En este apartado explicaremos de forma detallada y concisa los pasos a seguir para poder utilizar y manejar con la habilidad suficiente nuestra interfaz gráfica desarrollada para la mejor comprensión e interiorización de los conocimientos explicados en el tema de interpolación.

Para comenzar, abrimos nuestra interfaz gráfica, del mismo modo que anteriormente ha sido descrito, solo que en este caso seleccionaremos la correspondiente al tema de interpolación, denominada como “Tema5_Derivación e Integración”.

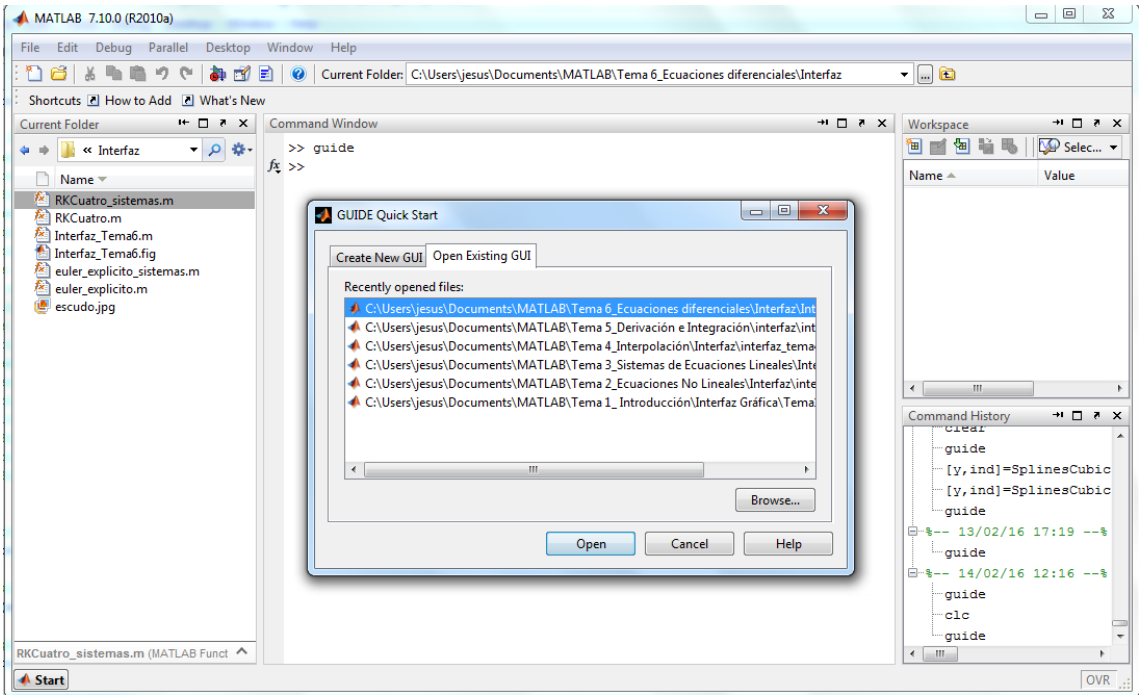


Figura 6.3

Al abrir dicho fichero aparecerá la interfaz de diseño de la propia interfaz gráfica, donde deberemos hacer doble click en el botón verde de reproducir o “play” situado en la zona superior.

Al hacerlo, aparecerá nuestra interfaz gráfica, inicialmente con la lista de métodos que han sido implementados y donde podremos seleccionar el método que queremos reproducir.

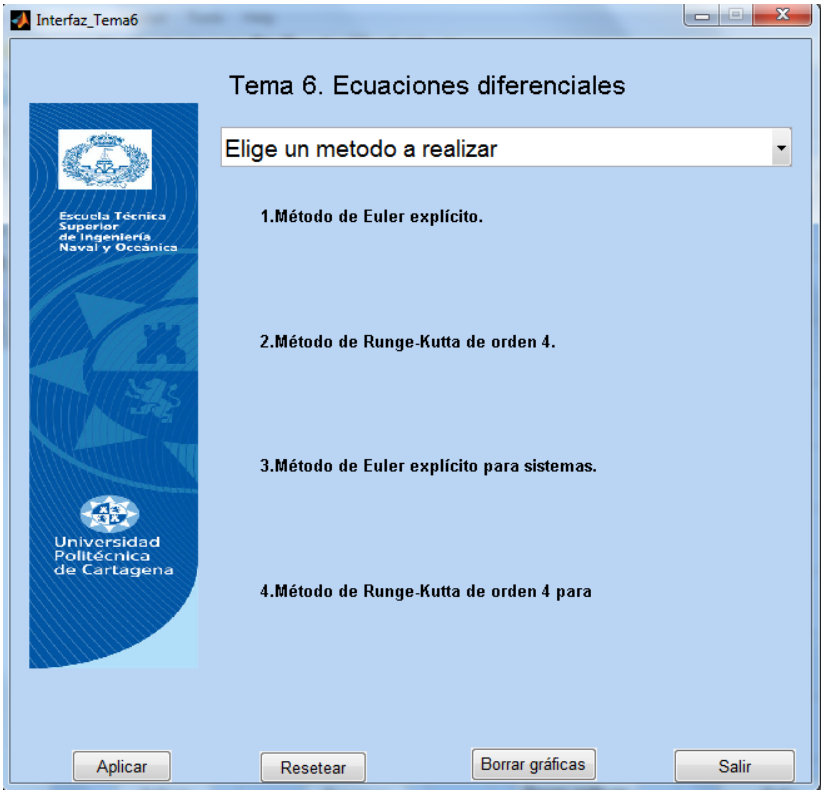


Figura 6.4

En este punto, pulsando en donde dice “Elige un método a realizar” nos aparece automáticamente una lista de los métodos de resolución de ecuaciones diferenciales ordinarias, donde elegiremos el que deseemos llevar a cabo.

1. Método de Euler explícito.

En primer lugar, si pulsáramos el botón denominado “Método de Euler explícito” nos aparece en pantalla un pdf con la teoría y ejemplos necesarios para comprender el método.

Probamos éste programa con ejemplo. Resolvemos o buscamos aproximaciones de las soluciones de la ecuación diferencial descrita por

$$\begin{cases} y'(t) = \dfrac{t-y}{2} \\ y(0) = 0 \end{cases}$$

En el intervalo $t \in [0\ 3]$, y con una distancia de tiempo entre cada iteración de $h = 1/2$.

El tamaño de paso entre cada iteración debe seleccionarse relativamente pequeño, ya que, de otro modo, se inducirían gran cantidad de errores en el método l basarse éste en la definición de derivada.

Para comenzar debemos introducir los datos en la interfaz;

Figura 6.5

Una vez introducidos los datos necesarios, al pulsar el botón aplicar aparece un archivo con los resultados obtenidos.

```
1  ********************************************************************************
2  Valores de entrada
3  ********************************************************************************
4  Expresión de la ecuación diferencial :
5  (t-y)/2
6  ********************************************************************************
7  Intervalo de aproximación:
8  [0.000  3.000]
9  ********************************************************************************
10 Valor inicial:
11 y(0.000)=0.000
12 ********************************************************************************
13 Paso en cada iteración:
14 0.500
15 ********************************************************************************
16 Resultados
17 ********************************************************************************
18 |    Abscisas    |  |       Y        |  |     Yexacto     |  |      error      |
19 |  0.000e+000    |  |  0.000e+000    |  |  0.000e+000     |  |  0.000e+000     |
20 |  5.000e-001    |  |  0.000e+000    |  |  5.760e-002     |  |  5.760e-002     |
21 |  1.000e+000    |  |  1.250e-001    |  |  2.131e-001     |  |  8.806e-002     |
22 |  1.500e+000    |  |  3.438e-001    |  |  4.447e-001     |  |  1.010e-001     |
23 |  2.000e+000    |  |  6.328e-001    |  |  7.358e-001     |  |  1.029e-001     |
24 |  2.500e+000    |  |  9.746e-001    |  |  1.073e+000     |  |  9.840e-002     |
25 |  3.000e+000    |  |  1.356e+000    |  |  1.446e+000     |  |  9.030e-002     |
```

Figura 6.6

Además con una representación gráfica de la expresión explícita obtenida como solución de la ecuación diferencial, y de las aproximaciones obtenidas mediante el método de Euler.

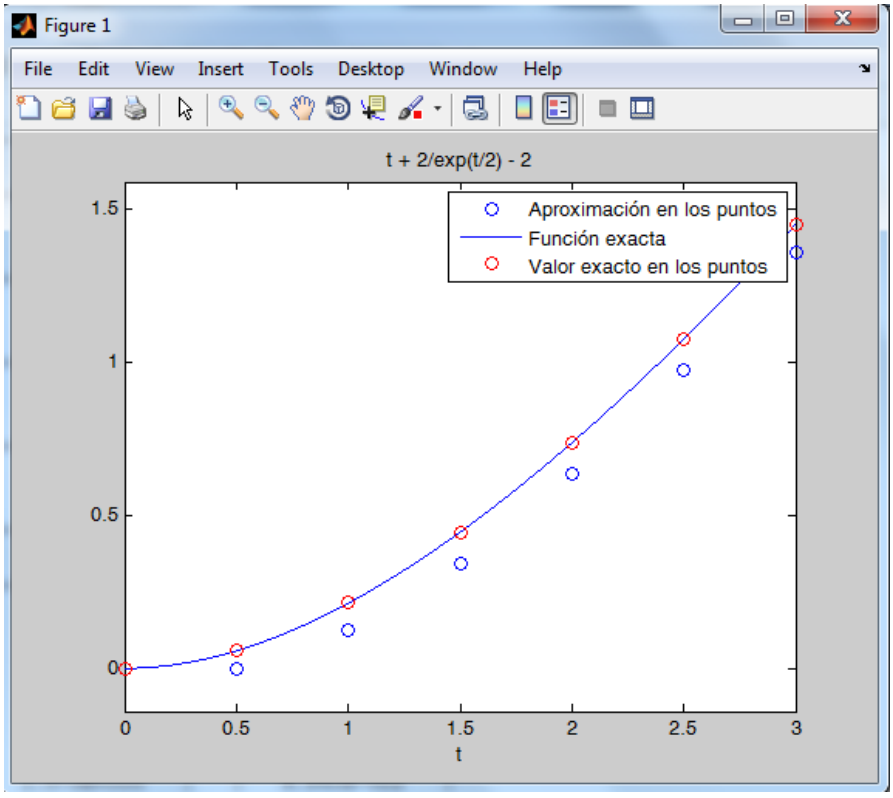


Figura 6.7

Podemos observar que las aproximaciones realizadas tienen bastante precisión, aunque se cometen errores. Como hemos señalado anteriormente, el cometido de errores aumenta de un modo exacerbado en éste método conforme aumentamos la distancia de tiempo entre las aproximaciones. Así, si aumentamos el paso entre iteraciones a $h = 2$,

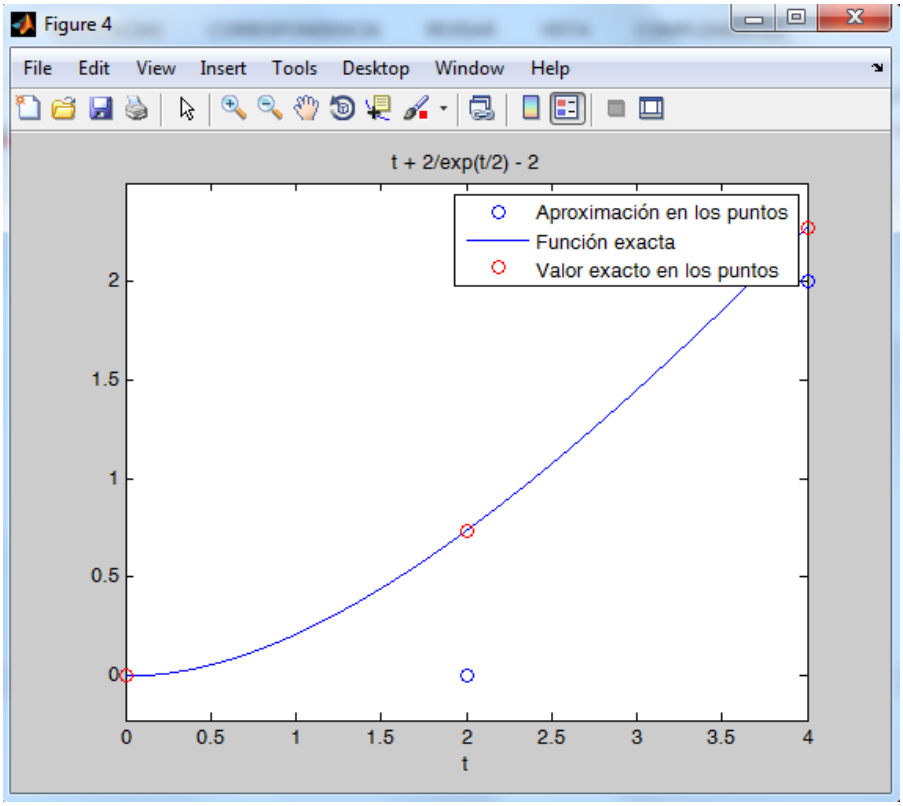


Figura 6.8

Vemos como el error cometido en la aproximación es notablemente superior.

2. Método de Runge-Kutta de orden 4.

En primer lugar, si pulsáramos el botón denominado “Método de Runge-Kutta” nos aparece en pantalla un pdf con la teoría y ejemplos necesarios para comprender el método.

Probamos éste programa con el mismo ejemplo que en el caso anterior para así poder observar la diferencia entre los resultaos obtenidos. Es decir, resolvemos la ecuación diferencial descrita por

$$\begin{cases} y'(t) = \dfrac{t-y}{2} \\ y(0) = 0 \end{cases}$$

En lugar de seleccionar un intervalo donde queremos que realice las aproximaciones, elegimos el valor inicial del tiempo, en este caso $t_0 = 0$ así como el número de iteraciones que deseamos que realice, en este caso $n = 4$ y con una distancia de tiempo entre cada iteración de $h = 1/2$.

En esta ocasión, también es recomendable elegir un tamaño de paso entre iteraciones relativamente pequeño para evitar mayores errores en la aproximación.

Para comenzar debemos introducir los datos necesarios en la interfaz gráfica.

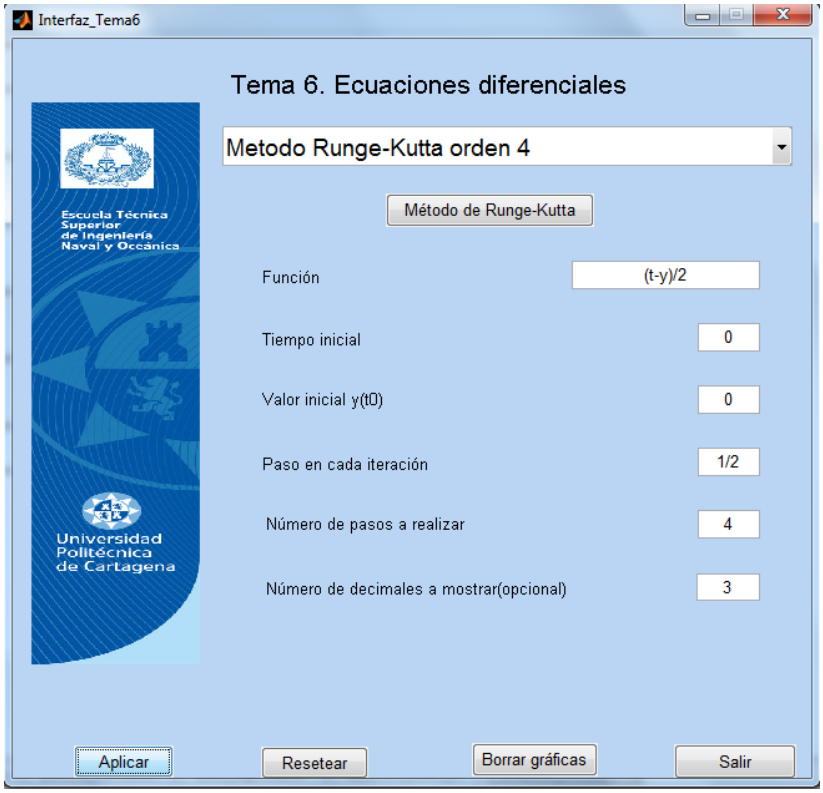


Figura 6.9

Una vez introducidos los datos necesarios, al pulsar el botón aplicar, aparece un archivo con los resultados obtenidos.

```
1  ************************************************************************
2  Valores de entrada
3  ************************************************************************
4  Expresión de la ecuación diferencial :
5  (t-y)/2
6  ************************************************************************
7  Valor del tiempo inicial:
8  0.000
9  ************************************************************************
10 Valor inicial:
11 y(0.000)=0.000
12 ************************************************************************
13 Paso de tiempo en cada iteración:
14 0.500
15 ************************************************************************
16 Número de pasos a realizar:
17 4.000
18 ************************************************************************
19 Resultados
20 ************************************************************************
21 |    Abscisas    |    |       Y       |    |    Yexacto    |    |      error      |
22 |   0.000e+000   |    |   0.000e+000   |    |   0.000e+000   |    |   0.000e+000   |
23 |   5.000e-001   |    |   5.762e-002   |    |   5.760e-002   |    |   -1.562e-005   |
24 |   1.000e+000   |    |   2.131e-001   |    |   2.131e-001   |    |   -2.433e-005   |
25 |   1.500e+000   |    |   4.448e-001   |    |   4.447e-001   |    |   -2.842e-005   |
26 |   2.000e+000   |    |   7.358e-001   |    |   7.358e-001   |    |   -2.952e-005   |
```

Figura 6.10

Además con una representación gráfica de la expresión explícita obtenida como solución de la ecuación diferencial, y de las aproximaciones obtenidas mediante el método de Runge-Kutta.

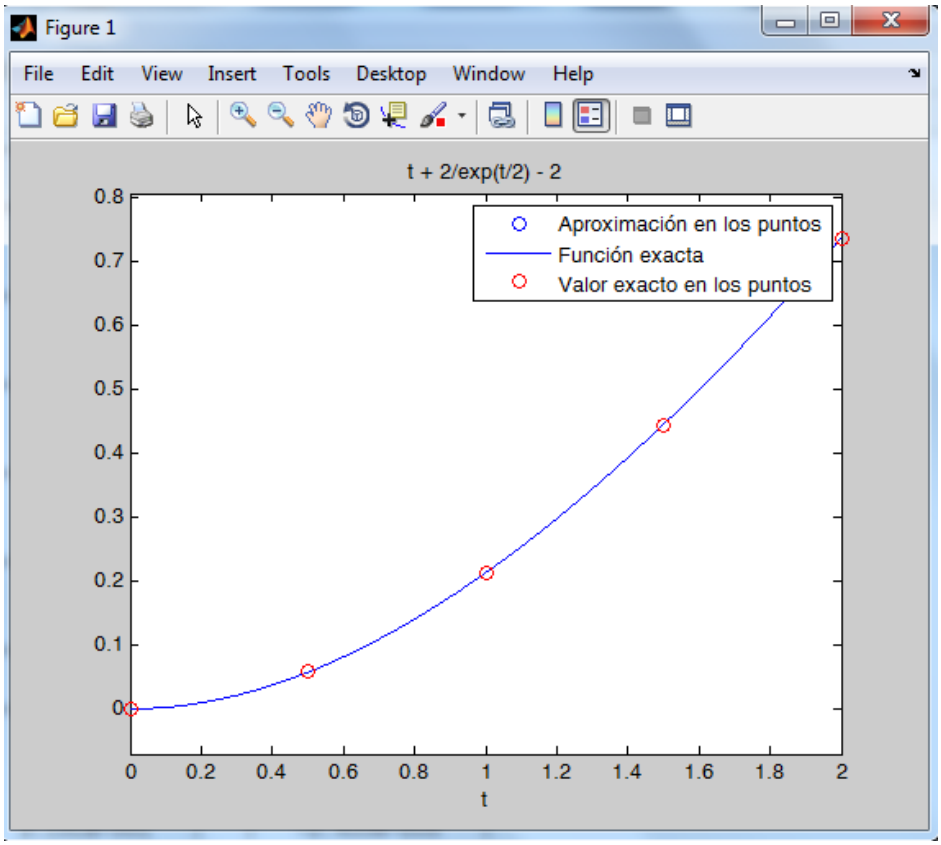


Figura 6.11

Como podemos observar, tanto en la representación gráfica como en los resultados obtenidos, este método aproxima con mayor exactitud que en el caso anterior, siendo en este caso el error cometido prácticamente nulo. Por lo que podemos observar como éste es más preciso que el método anterior, tal y como anunciamos y demostramos en teoría.

Sin embargo, el grado de aproximación de las soluciones depende además de la ecuación diferencial a resolver. Por ejemplo si tratamos de obtener soluciones aproximadas para la ecuación definida por,

$$\begin{cases} y'(t) = e^{-2t} - 2y \\ y(0) = 0.1 \end{cases}$$

Siendo $t_0 = 0$ y realizaremos un total de dos iteraciones a una distancia de $h = 0.8$ entre ellas.

Una vez introducimos los datos en la interfaz gráfica, pulsamos el botón aplicar y observamos los resultados obtenidos.

```
1  ********************************************************************************
2  Valores de entrada
3  ********************************************************************************
4  Expresión de la ecuación diferencial :
5  exp(-2*t)-2*y
6  ********************************************************************************
7  Valor del tiempo inicial:
8  0.000
9  ********************************************************************************
10 Valor inicial:
11 y(0.000)=0.100
12 ********************************************************************************
13 Paso de tiempo en cada iteración:
14 0.800
15 ********************************************************************************
16 Número de pasos a realizar:
17 2.000
18 ********************************************************************************
19 Resultados
20 ********************************************************************************
21 |    Abscisas    |  |       Y        |  |    Yexacto     |  |      error     |
22 |  0.000e+000    |  |  1.000e-001    |  |  1.000e-001    |  |  0.000e+000    |
23 |  8.000e-001    |  |  1.327e-001    |  |  1.817e-001    |  |  4.900e-002    |
24 |  1.600e+000    |  |  5.722e-002    |  |  6.930e-002    |  |  1.208e-002    |
```

Figura 6.12

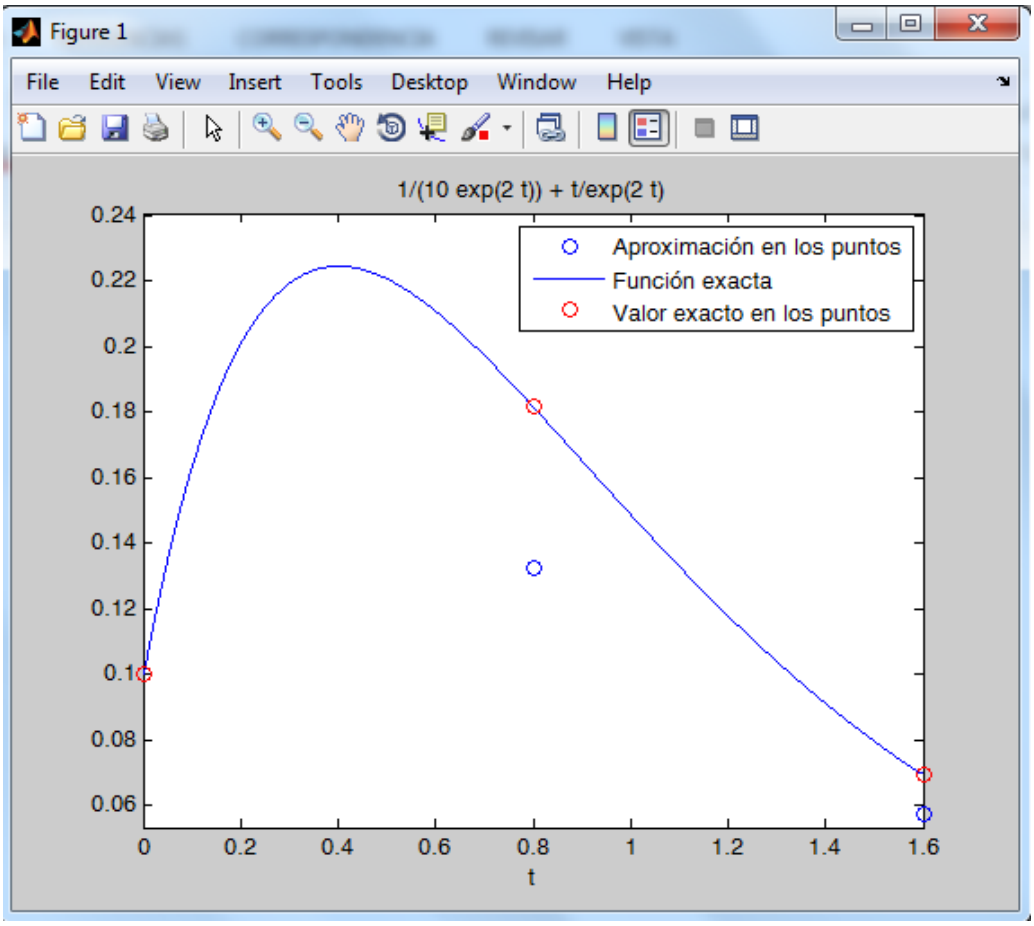


Figura 6.13

Tal y como podemos observar tanto en la representación gráfica como en los resultados, al contrario que en el caso anterior, se cometen errores notables en la aproximación de soluciones.

Por otro lado, también existen casos para los que no existe una solución explícita de la ecuación diferencial. En estos casos el programa se limita a realizar las aproximaciones, las cuales no podemos saber de forma exacta su error con la realidad.

3. Método de Euler explícito para sistemas.

En primer lugar, si pulsáramos el botón denominado "Método de Euler para sistemas" nos aparece en pantalla un pdf con la teoría y ejemplos necesarios para comprender el método.

Probaremos este programa con el sistema de ecuaciones diferenciales definido por

$$\begin{cases} x' = x + y - 7 \\ y' = 3x + 2y + 5 \end{cases}$$

Cumpliendo con las condiciones iniciales $x(0) = 2\ e\ y(0) = 3$ en el intervalo $[0\ 1]$ con un paso de discretización de $h = 0.2$.

Para comenzar debemos introducir los datos necesarios en la interfaz gráfica.

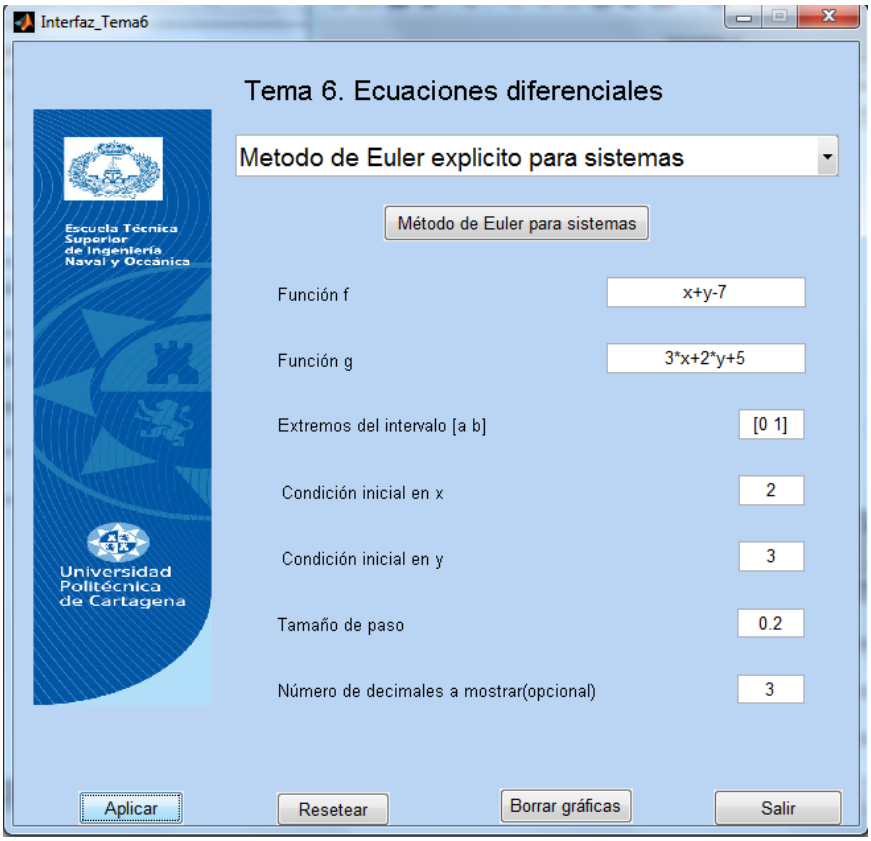


Figura 6.14

Una vez que hemos introducido los datos necesarios, al pulsar el botón aplicar apare un archivo con los resultados obtenidos.

```
Expresión de la ecuación diferencial f:
x+y-7
********************************************************************************
Expresión de la ecuación diferencial g:
3*x+2*y+5
Intervalo de aproximación:
[0.000  1.000]
********************************************************************************
Condición inicial de la variable x:
x(0.000)= 2.000
********************************************************************************
Condición inicial de la variable y:
y(0.000)= 3.000
Tamaño de paso en cada iteración
0.200
********************************************************************************
Resultados
********************************************************************************
|    Abscisas    |  |       X        |  |       Y        |  |     Xexacto    |  |    Yexacto     |  |    errorx      |  |    errory      |
|  0.000e+000    |  |  2.000e+000    |  |  3.000e+000    |  |  2.000e+000    |  |  3.000e+000    |  |  0.000e+000    |  |  0.000e+000    |

|  2.000e-001    |  |  1.600e+000    |  |  6.400e+000    |  |  1.968e+000    |  |  7.120e+000    |  |  3.684e-001    |  |  7.200e-001    |

|  4.000e-001    |  |  1.800e+000    |  |  1.092e+001    |  |  3.064e+000    |  |  1.359e+001    |  |  1.264e+000    |  |  2.669e+000    |

|  6.000e-001    |  |  2.944e+000    |  |  1.737e+001    |  |  6.273e+000    |  |  2.469e+001    |  |  3.329e+000    |  |  7.326e+000    |

|  8.000e-001    |  |  5.606e+000    |  |  2.708e+001    |  |  1.351e+001    |  |  4.486e+001    |  |  7.904e+000    |  |  1.777e+001    |

|  1.000e+000    |  |  1.074e+001    |  |  4.228e+001    |  |  2.849e+001    |  |  8.263e+001    |  |  1.774e+001    |  |  4.035e+001    |
```

Figura 6.15

Además de las presentaciones gráficas de los valores aproximados y exactos obtenidos tanto en la variable x como y;

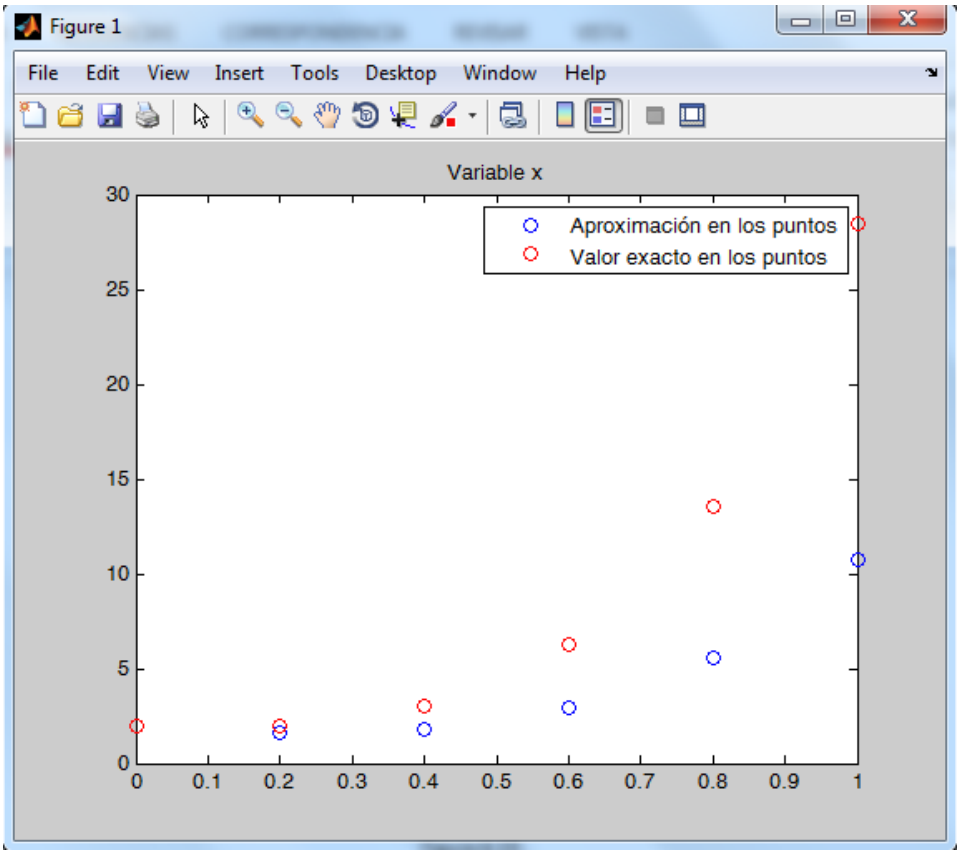


Figura 6.16

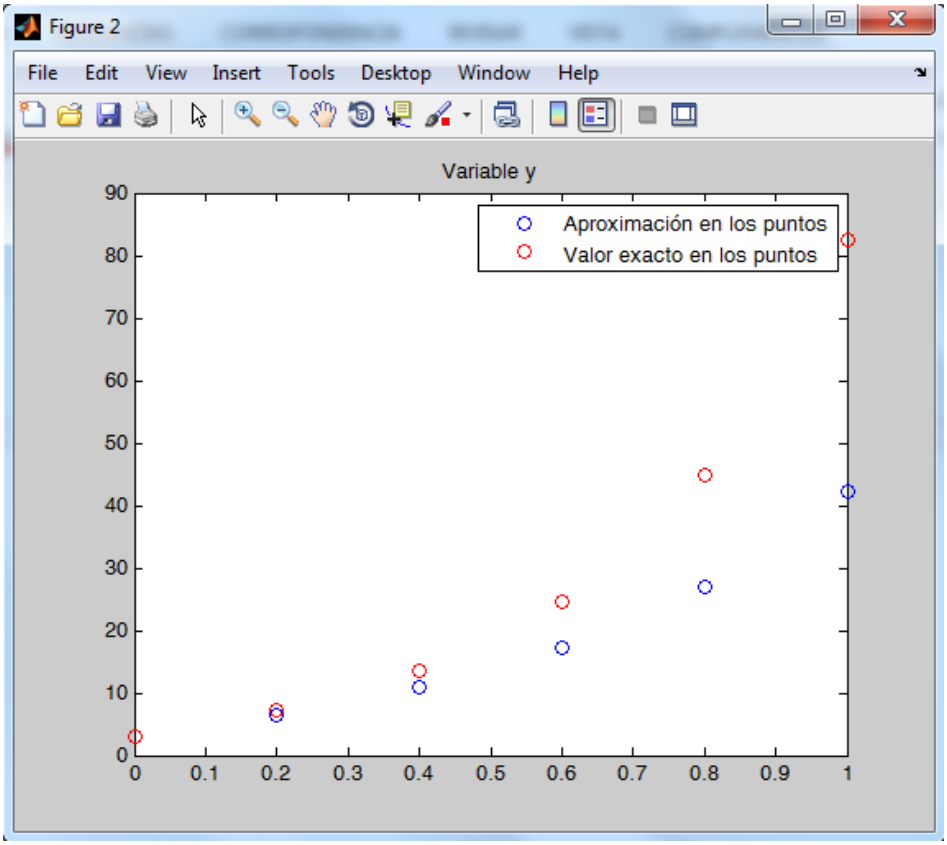


Figura 6.17

Como podemos observar, conseguimos aproximaciones bastantes precisas para la solución de ambas variables. Sin embargo, el error cometido en ambas se agranda conforme avanzamos en el tiempo.

Existen, además, algunos sistemas de ecuaciones diferenciales para los que no existe una solución exacta o explicita del sistema. Por ejemplo, tratamos de resolver el sistema compuesto por

$$\begin{cases} x' = x * y + t \\ y' = x - t \end{cases}$$

Cumpliendo las condiciones iniciales $x(0) = 0, y(0) = 1$ en el intervalo $[0\ 1]$ para un paso de discretización $h = 0.2$.

Una vez introducidos los datos en nuestra interfaz gráfica, al pulsar el botón aceptar aparece un archivo con los resultados obtenidos, que tan sólo contiene las aproximaciones realizadas ya que no es posible hacer el cálculo de manera exacta.

Archivo Edición Formato Ver Ayuda

```
**********************************************************************************
Valores de entrada
**********************************************************************************
Expresión de la ecuación diferencial f:
x*y+t
**********************************************************************************
Expresión de la ecuación diferencial g:
x-t
Intervalo de aproximación:
[0.000  1.000]
**********************************************************************************
Condición inicial de la variable x:
x(0.000)= 0.000
**********************************************************************************
Condición inicial de la variable y:
y(0.000)= 1.000
Tamaño de paso en cada iteración
0.200
**********************************************************************************
Resultados
**********************************************************************************
No existe uns expresión explicita para la función
|     Abscisas     |     X     |     Y     |
|   0.000e+000     |   0.000e+000     |   1.000e+000     |
|   2.000e-001     |   0.000e+000     |   1.000e+000     |
|   4.000e-001     |   4.000e-002     |   9.600e-001     |
|   6.000e-001     |   1.277e-001     |   8.880e-001     |
|   8.000e-001     |   2.704e-001     |   7.935e-001     |
|   1.000e+000     |   4.733e-001     |   6.876e-001     |
```

Figura 6.18

Además de una representación gráfica de las aproximaciones obtenidas tanto de la variable x como y;

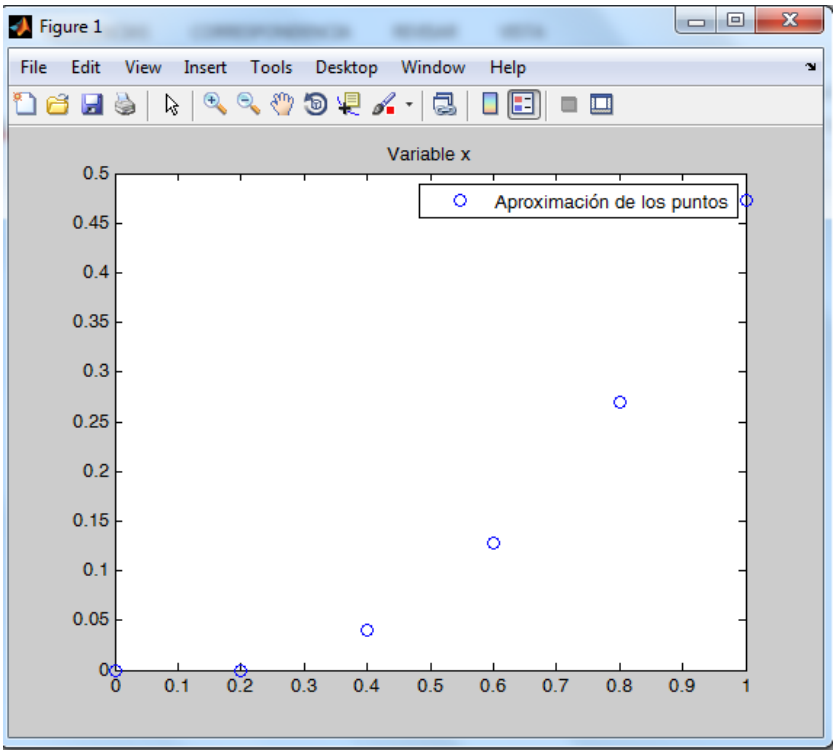


Figura 6.19

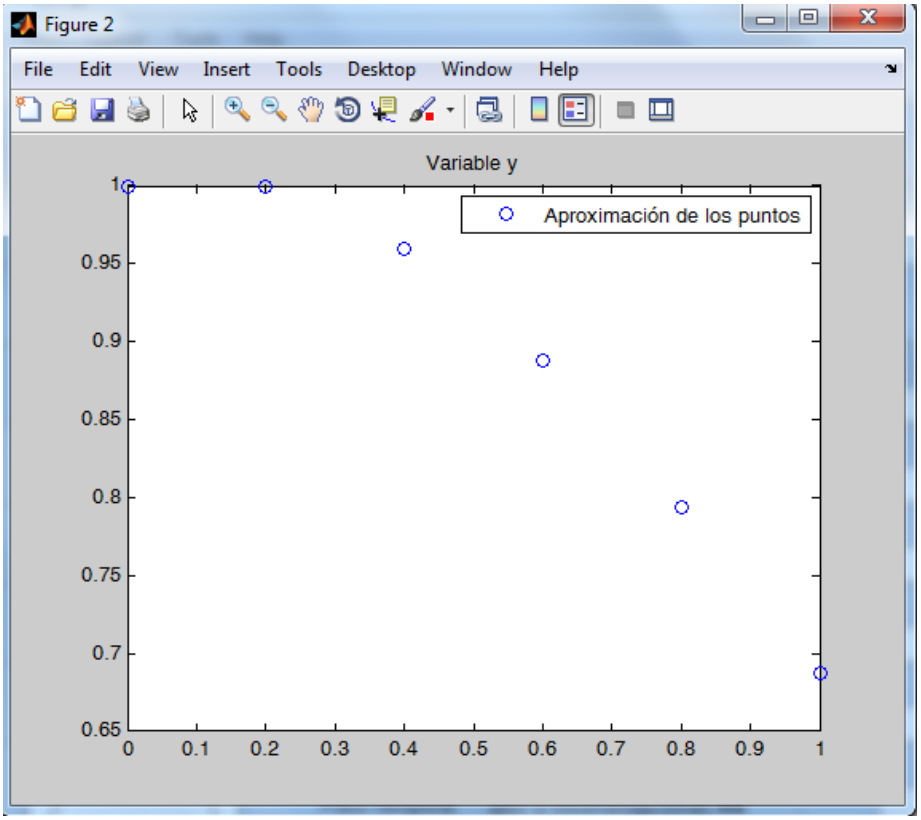


Figura 6.20

4. Método de Runge-Kutta de orden 4 para sistemas.

En primer lugar, si pulsáramos el botón denominado “Método de Runge-Kutta para sistemas” nos aparece en pantalla un pdf con la teoría y ejemplos necesarios para comprender el método.

Probaremos este programa con el mismo ejemplo utilizado en el caso anterior para así poder observar la diferencia en los resultados obtenidos, es decir, tratamos de resolver el sistema de ecuaciones diferenciales definido por

$$\begin{cases} x' = x + y - 7 \\ y' = 3x + 2y + 5 \end{cases}$$

Cumpliendo con las condiciones iniciales $x(0) = 2\ e\ y(0) = 3$. Al igual que en el método de Runge-Kutta para ecuaciones aquí debemos introducir el tiempo inicial, que en este caso será $t_0 = 0$, el número de iteraciones que queremos realizar, que en este caso haremos tres, y el paso de discretización al que deseamos realizar dichas iteraciones, en este caso $h = 0.2$.

Introducimos dichos datos en la interfaz gráfica,

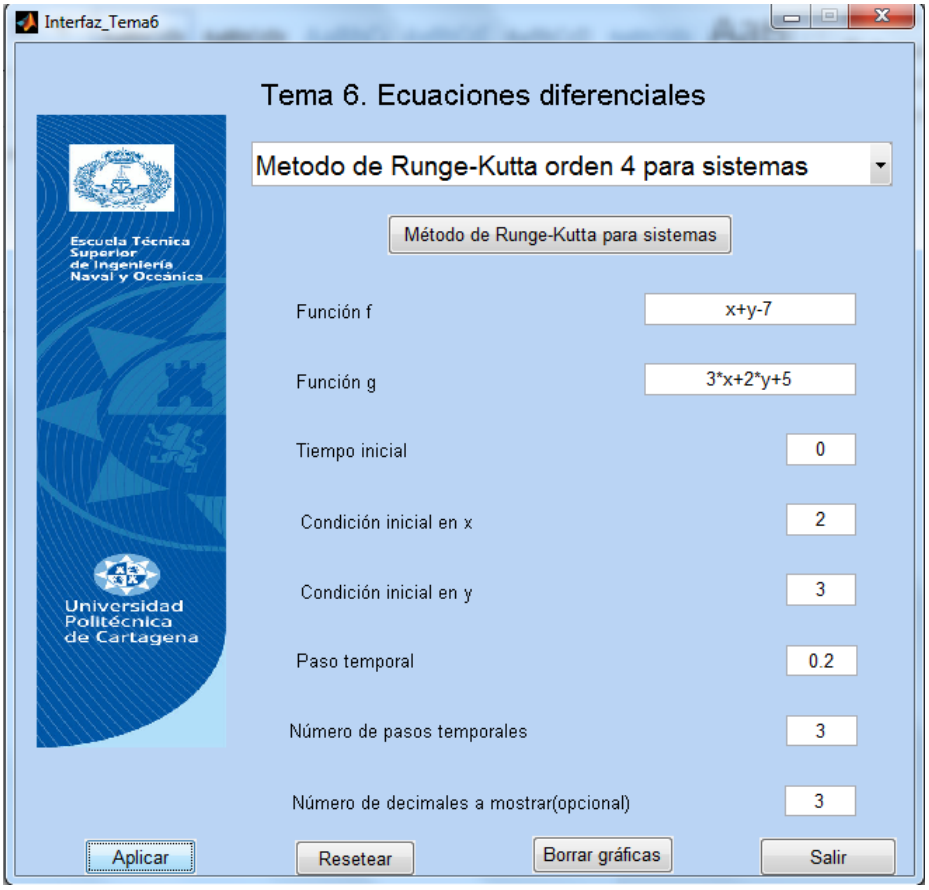


Figura 6.21

Una vez introducidos los datos necesarios, al pulsar el botón aplicar aparece un archivo con los resultados obtenidos;

```
**************************************************************************
Valores de entrada
**************************************************************************
Expresión de la ecuación diferencial f:
x+y-7
**************************************************************************
Expresión de la ecuación diferencial g:
3*x+2*y+5
Tiempo inicial:
0.000
**************************************************************************
Condición inicial de la variable x:
x(0.000)= 2.000
**************************************************************************
Condición inicial de la variable y:
y(0.000)= 3.000
Tamaño de paso en cada iteración
0.200
**************************************************************************
Número de pasos temporales a realizar:
3.000
Resultados
**************************************************************************
|    Abscisas    |  |       X       |  |       Y       |  |    Xexacto    |  |    Yexacto    |  |    errorx     |  |    errory     |
|  0.000e+000    |  |  2.000e+000   |  |  3.000e+000   |  |  2.000e+000   |  |  3.000e+000   |  |  0.000e+000   |  |  0.000e+000   |

|  2.000e-001    |  |  1.967e+000   |  |  7.117e+000   |  |  1.968e+000   |  |  7.120e+000   |  |  1.420e-003   |  |  3.271e-003   |

|  4.000e-001    |  |  3.058e+000   |  |  1.358e+001   |  |  3.064e+000   |  |  1.359e+001   |  |  5.498e-003   |  |  1.266e-002   |

|  6.000e-001    |  |  6.257e+000   |  |  2.466e+001   |  |  6.273e+000   |  |  2.469e+001   |  |  1.596e-002   |  |  3.676e-002   |
```

Figura 6.22

Además de las presentaciones gráficas de los valores aproximados y exactos obtenidos tanto en la variable x como y;

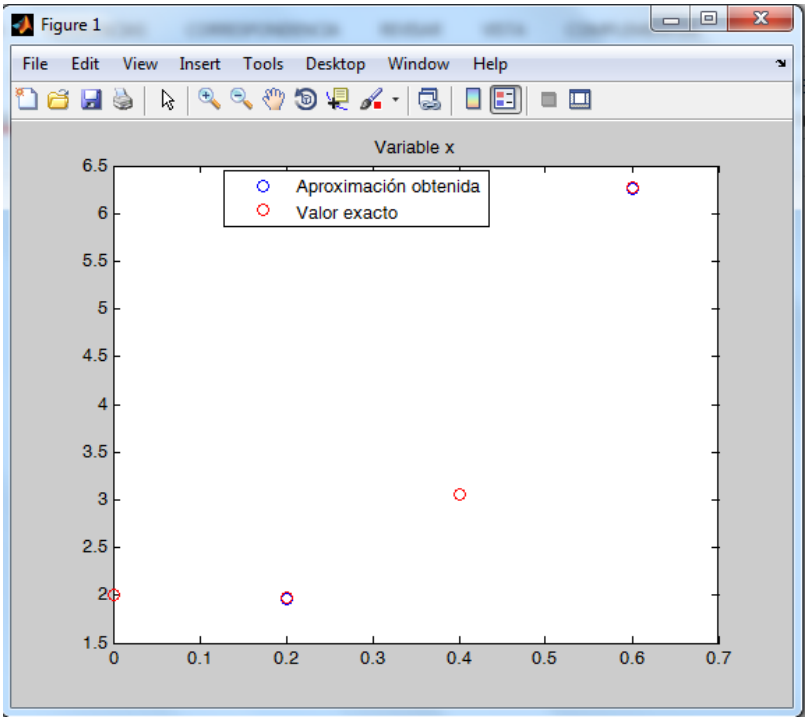


Figura 6.23

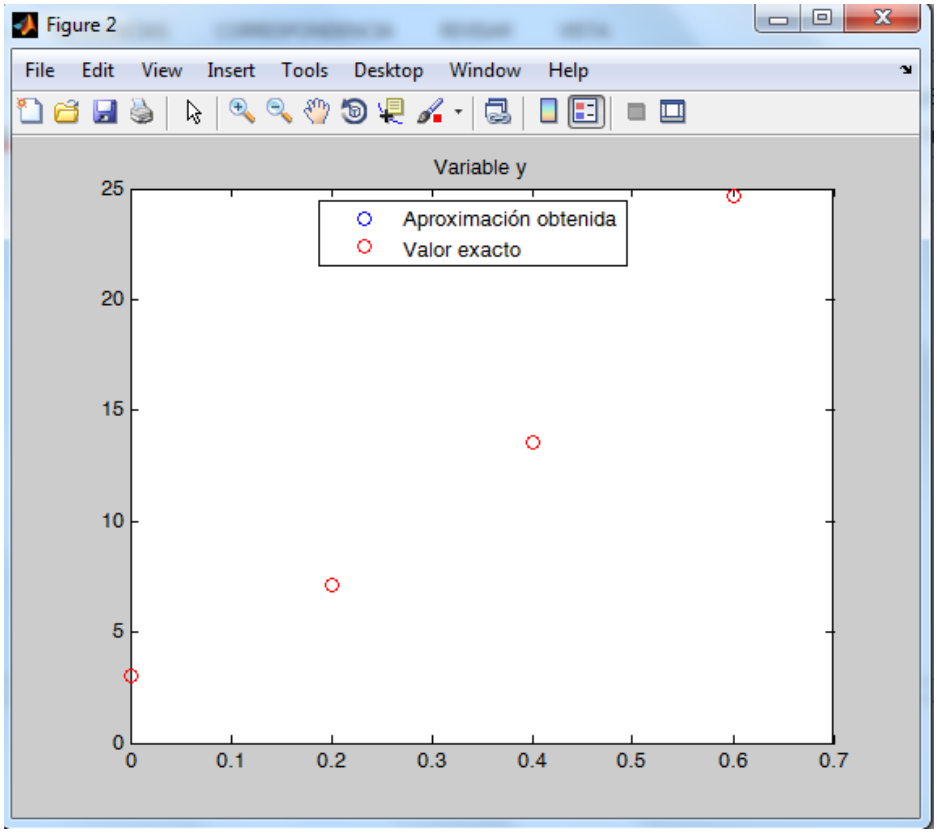


Figura 6.24

Como podemos observar, tanto en los resultados obtenidos como en las representaciones gráficas, las aproximaciones realizadas son mucho más precisas que las obtenidas mediante el método de Euler, no llegando incluso a apreciar a simple vista las diferencias o el error cometido en las representaciones gráficas de ambas variables.

Esto confirma lo enunciado y descrito en teoría.

Por otro lado, al igual que en el caso anterior, puede darse el caso de que no exista una solución explícita del sistema. Realizamos el mismo ejemplo del caso anterior para este fenómeno.

Tratamos de resolver el sistema compuesto por;

$$\begin{cases} x' = x * y + t \\ y' = x - t \end{cases}$$

Cumpliendo las condiciones iniciales $x(0) = 0, y(0) = 1$, comenzando en $t_0 = 0$, realizando un total de tres iteraciones a un paso de discretización $h = 0.2$.

Una vez que hemos introducido los datos necesarios en la interfaz gráfica, pulsamos el botón aceptar y aparece un archivo con los resultados de las aproximaciones obtenidas.

```
*******************************************************************************
Valores de entrada
*******************************************************************************
Expresión de la ecuación diferencial f:
x*y+t
*******************************************************************************
Expresión de la ecuación diferencial g:
x-t
Tiempo inicial:
0.000
*******************************************************************************
Condición inicial de la variable x:
x(0.000)= 0.000
*******************************************************************************
Condición inicial de la variable y:
y(0.000)= 1.000
Tamaño de paso en cada iteración
0.200
*******************************************************************************
Número de pasos temporales a realizar:
3.000
Resultados
*******************************************************************************
No existe uns expresión explicita para la función
|     Abscisas    |    |        X        |    |        Y        |
|   0.000e+000    |    |   0.000e+000    |    |   1.000e+000    |

|   2.000e-001    |    |   2.138e-002    |    |   9.814e-001    |

|   4.000e-001    |    |   9.127e-002    |    |   9.318e-001    |

|   6.000e-001    |    |   2.180e-001    |    |   8.617e-001    |
```

Figura 6.25

Además de una representación gráfica de las aproximaciones obtenidas tanto de la variable x como y;

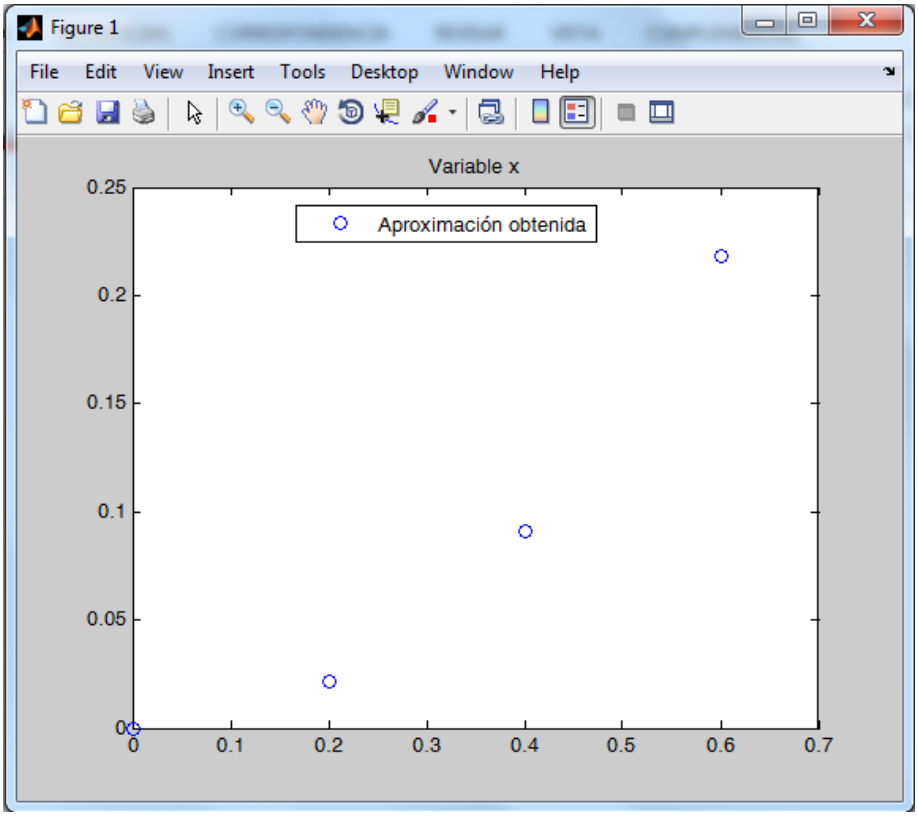


Figura 6.26

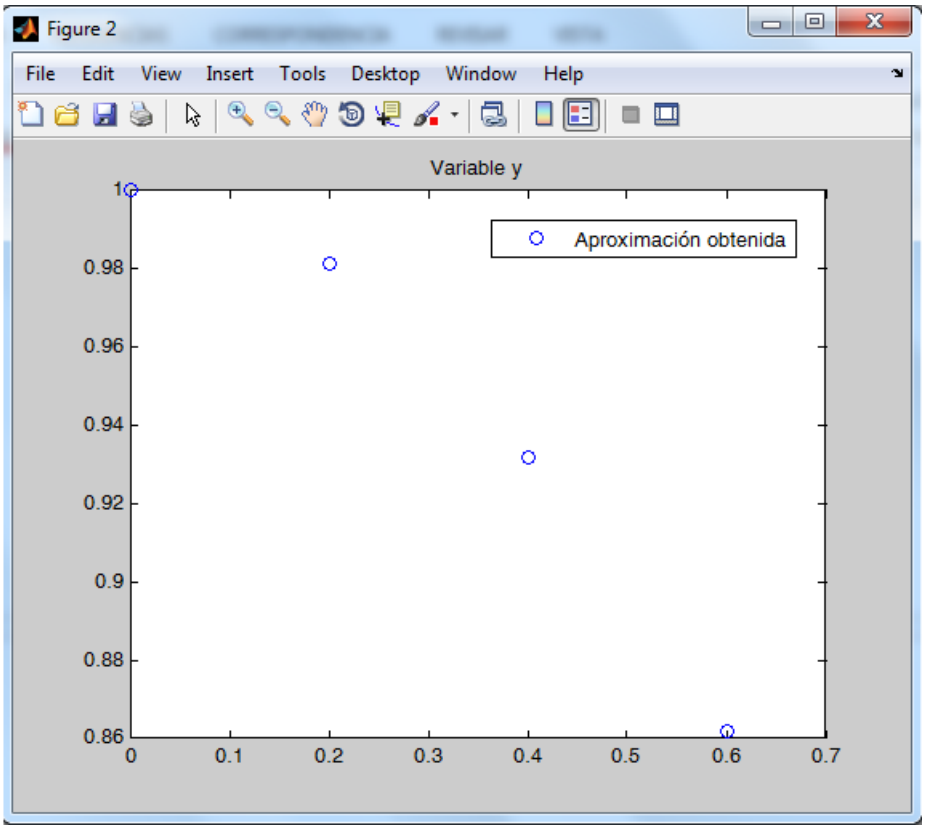


Figura 6.27

A pesar de que no existe una manera de comprobar el error exacto cometido en las aproximaciones de ambas variables, deducimos que éstas tienen mayor precisión que las obtenidas mediante el método e Euler ya que, como hemos demostrado, el método de Runge-Kutta tiene una mayor precisión que este.

Anexo I.

Programas implementados en código Matlab.

Capítulo I.

Algoritmo inestable.

```
function [im,ib,error_abs]=ejemplo1_estabilidad(a,n)

% En este programa comparamos dos posibles algoritmos
% para resolver la integral In=int(x^n/(a+x),0,1)
% Un algoritmo no estable para |a|>1 que utiliza la
% recurrencia I_n=1/n-aI_{n-1}
% Un algoritmo bueno basado en integración numérica
% [im,ib,error_abs]=ejemplo1_estabilidad(a,n)
% Variables de entrada:
% a,n parámetros de la integral
% Variables de salida:
% im valor de la integral calculado por el método recursivo
% ib valor de la integral calculado con el método de
%    integración numérica
% error_abs  error absoluto entre ib y im
% Ejemplo:
% [im,ib,error_abs]=ejemplo1_estabilidad(11,10)

% definimos unas variables simbólicas
syms x as ns;

% calculamos el valor de la integral por el método inestable
I0=log((1+a)/a);
In=zeros(n,1);
In(1)=I0;
for i=2:n
    im=1/i-a*I0;
    I0=im;
    In(i)=I0;
end
In
% calculamos el valor de la integral por el método estable basado
% en integración numérica

f='x^ns/(as+x)';
ib=double(int(subs(f,{ns,as},{n,a}),0,1));


% calculamos el error absoluto entre las dos aproximaciones

error_abs=abs(im-ib);
```

Epsilon máquina.

```
% Cálculo de epsilon máquina

x=1;
while 1+x>1
   x=x/2;
end
x=2*x;
disp('El épsilon de la máquina es')
disp(x)
```

Error de la división por 10.

```
function [s,sex,ErrorAbsoluto,ErrorRelativo]=suma_redondeo(n)

% Esta función muestra la acumulación de errores de redondeo al
% sumar n veces la fracción 1/10
% [s,sex,error]=suma_redondeo(n);
% Variables de entrada:
% n número de sumandos en la suma
% Variables de salida:
% s valor calculado para la suma
% sex suma exacta que vale n/10
% error cometido al aproximar la suma exacta por la aproximación
% Ejemplo:
% [s,sex,error]=suma_redondeo(1000)


s=0;
for i=1:n
    s=s+1/10;
end

sex=n/10;
ErrorAbsoluto=abs(sex-s);
ErrorRelativo=ErrorAbsoluto/sex;
```

Suma de la serie de inversos cuadrados.

```
function [sp,sr,sinf,errores]=suma_inversos_cuadrados(n)

% Este programa calcula la suma de los inversos de los cuadrados desde
0 hasta n de r^n
% [sp,sr,sinf,errores]=suma_inversos_cuadrados(n);
% Variables de entrada:
% n número de terminos en la serie
% Variables de salida:
% sp suma hecha de manera progresiva desde el termino mayor en valor
%    absoluto al menor
% sr suma hecha de manera regresiva desde el termino de menor valor
%    absoluto
```

```
% sinf suma exacta de la serie, que en este caso es conocido y vale
pi^2/6
% errores es un vector que contiene dos componentes. La primera es el
error
%         entre la suma infinita exacta y la aproximación progresiva y
la
%         segunda el error entre la suma infinita exacta y la
aproximación
%         regresiva, es decir
%
%         errores(1)=sinf-sp
%         errores(2)=sinf-sr
% Ejemplo:
% [sp,sr,sinf,errores]=suma_inversos_cuadrados(1000)


% Suma progresiva
sp=1;
for i=1:n
    sp=sp+1/i^2;
end


% Suma regresiva
sr=1/n^2;
for i=n-1:-1:1
    sr=sr+1/i^2;
end


% Cálculo de errores
sinf=pi^2/6;
errores(1)=sinf-sp;
errores(2)=sinf-sr;
```

Desarrollo de Taylor.

```
function [t,tnum,fxc,txc,e]=desarrollo_taylor(f,x0,n,xc)

% Este programa da el desarrollo en serie de Taylor de una funcion
introducida
% como una cadena de caracteres. Así mismo evalúa la aproximación y la
% función en el punto de abscisa introducido
% [t,fxc,txc,e]=desarrollo_taylor(f,x0,n,xc);
% Variables de entrada:
% f es la cadena de caracteres que contiene la función
% x0 es el punto alrededor del cual hacemos el desarrollo
% n es el grado del desarrollo de Taylor que se desea
% xc es el punto en el que se desea evaluar la función y la
aproximación
% Variables de salida:
% t polinomio de Taylor
% fxc evaluación de la función f en el punto xc
% txc evaluación de la aproximación en el punto xc
% e error en la aproximación fxc-txc
% Ejemplo:
% [t,tnum,fxc,txc,e]=desarrollo_taylor('sin(x)',0,5,1)
```

```
% Cálculo simbólico del desarrollo de Taylor

syms x

t=subs(f,x0);
tnum=num2str(double(subs(f,x0)));
factorial=1;
for i=1:n
    factorial=factorial*i;
    dt=((subs(diff(f,i),x0)))/(factorial)*((x-x0)^(i));
    dtnum=double(subs(diff(f,i),x0)/factorial);
    t=t+dt;
    if dtnum>0
        if x0>0
            tnum=[tnum,'+',num2str(dtnum),'(x-
',num2str(x0),')^',num2str(i)];
        elseif x0<0
            tnum=[tnum,'+',num2str(dtnum),'(x+',num2str(-
x0),')^',num2str(i)];
        else
            tnum=[tnum,'+',num2str(dtnum),'x^',num2str(i)];
        end

    elseif dtnum<0
        if x0>0
            tnum=[tnum,num2str(dtnum),'(x-
',num2str(x0),')^',num2str(i)];
        elseif x0<0
            tnum=[tnum,num2str(dtnum),'(x+',num2str(-
x0),')^',num2str(i)];
        else
            tnum=[tnum,num2str(dtnum),'x^',num2str(i)];
        end
    end

end

% Evaluación de la función y del polinomio de Taylor en xc

txc=subs(t,xc);
fxc=subs(f,xc);

% Error en la aproximación
e=fxc-txc;
```

Series de Fourier.

```
function [sn,snum]=fourier_general(f,P,n,I)

% Esta función calcula el desarrollo de la suma parcial de la serie de
Fourier
% para un número finito n de frecuencias
% [sn,sunum]=fourier_general(f,P,n,I)
% Variables de entrada:
% f es la función que queremos desarrollar en serie de Fourier
%    DEBE ser introducida como función HANDLE
```

```
% P es la mitad del periodo 2P de la función, es decir, debe estar
definida en [-P,P]
% n es el grado de aproximación que se desea
% I vector de dos componentes que indican el número de copias
periódicas a la izquierda
%   y a la derecha para dibujar la aproximación
% Variables de salida:
% sn es el desarrollo de la serie de Fourier de la función dada
% snum es el desarrollo de la serie de Fourier con coeficientes
aproximados
%      numéricamente
% Ejemplo:
% [sn,snum]=fourier_general(@(x)x,pi,20,[1,1])

% inicializaciones
syms  x
a=double(zeros(n,1));
b=double(zeros(n,1));

% Cálculo del termino independiente

a0=(quad(f,-P,P))/P;
snum=num2str(a0);

% añadimos el resto de terminos

sn=a0;
handle_coseno=@(fre,per,x)f(x).*cos(fre*pi*x/per);
handle_seno=@(fre,per,x)f(x).*sin(fre*pi*x/per);
for k=1:n
    coeficiente_coseno=@(x)handle_coseno(k,P,x);
    a(k)=quad(coeficiente_coseno,-P,P)/P;

    coeficiente_seno=@(x)handle_seno(k,P,x);
    b(k)=quad(coeficiente_seno,-P,P)/P;

    sn=sn+a(k)*cos(k*pi/P*x)+b(k)*sin(k*pi/P*x);
   if a(k)>0
       snum=[snum,'+',num2str(a(k)),'*cos(',num2str(k*pi/P),'*x)'];
   elseif a(k)<0
       snum=[snum,num2str(a(k)),'*cos(',num2str(k*pi/P),'*x)'];
   end

   if b(k)>0
       snum=[snum,'+',num2str(b(k)),'*sin(',num2str(k*pi/P),'*x)'];
   elseif b(k)<0
       snum=[snum,num2str(b(k)),'*sin(',num2str(k*pi/P),'*x)'];
   end

end


% dibujo de la función y de la aproximación

h=1/100;

ext_izq=-P*(2*I(1)+1);
ext_der=P*(2*I(2)+1);
taux=ext_izq:h:ext_izq+2*P;
```

```
faux=subs(f,{x},{taux+2*P*I(1)});
t=taux;
ft=faux;
ext_izq_aux=ext_izq+2*P;

for copias=1:I(1)+I(2)
    taux=ext_izq_aux+h:h:ext_izq_aux+2*P;
    faux=subs(f,{x},{taux+(2*P*I(1)-copias*2*P)});
    ft=[ft,faux];
    t=[t,taux];
    ext_izq_aux=ext_izq_aux+2*P;
end

yt=subs(sn,{x},{t});

M=max(ft);
m=min(ft);
margenV=(M-m)/10;
margenH=(ext_der-ext_izq)/10;


axis([ext_izq-margenH ext_der+margenH m-margenV M+margenV])
h1=plot(t,ft,'b','LineWidth',3);
hold on
h2=plot(t,yt,'r','LineWidth',2);
legend([h1,h2],'Función',['Serie de fourier con ',num2str(n),'
frecuencias'])
```

Error discretización primera derivada.

```
function primera_derivada_error_discretizacion(f,c,h)

% Esta función aplica las reglas de aproximación de la primera
derivada
% usuales y calcula el error cometido en la aproximación, que se llama
% error de discretización
% primera_derivada_error_discretizacion(f,c,h)
% Variables de entrada:
% f función como cadena de caracteres
% c punto donde queremos aproximar la derivada
% h paso de discretización
% Ejemplo:
%  primera_derivada_error_discretizacion('cos(x)',1,1/10)

% Derivada exacta en c
df=diff(f);
dfc=subs(df,c);

% Aproximación por la derivada progresiva

dpro=(subs(f,c+h)-subs(f,c))/h;
fprintf('Error de discretización en la fórmula progresiva = %f\n',dfc-
dpro);

% Aproximación por la derivada regresiva

dreg=(subs(f,c)-subs(f,c-h))/h;
```

```
fprintf('Error de discretización en la fórmula regresiva = %f\n',dfc-
dreg);

% Aproximación por la derivada centrada

dcen=(subs(f,c+h)-subs(f,c-h))/(2*h);
fprintf('Error de discretización en la fórmula centrada = %f\n',dfc-
dcen);
```

Error discretización segunda derivada.

```
function segunda_derivada_error_discretizacion(f,c,h)

% Esta función aplica la regla de aproximación de la segunda derivada
% más usual (f(c+h)-2f(c)+f(c-h))/h^2 y calcula el error cometido en
la
% aproximación, que se llama error de discretización
% segunda_derivada_error_discretizacion(f,c,h)
% Variables de entrada:
% f función como cadena de caracteres
% c punto donde queremos aproximar la derivada
% h paso de discretización
% Ejemplo:
%  segunda_derivada_error_discretizacion('cos(x)',1,1/10)

% Derivada exacta en c
d2f=diff(f,2);
d2fc=subs(d2f,c);

% Aproximación de la segunda derivada

d2apro=(subs(f,c+h)-2*subs(f,c)+subs(f,c-h))/h^2;
error=d2fc-d2apro;
if error>0
    fprintf('Aproximación por defecto\n');
elseif error<0
    fprintf('Aproximación por exceso\n');
end

fprintf('Error de discretización en la fórmula para la segunda
derivada = %f\n',error);
```

Propagación directa de errores.

```
function [r,dr,intervalo]=propagacion_directa_errores(f,x,dx)

% Esta función ejemplifica la propagación de errores con
% la función de n (n<=10) variables f(x1,x2,...,xn) donde las
variables xi han
% sido medidas en el vector x con errores contenidos en el vector dx.
% Las variables deben de llamarse con la nomenclatura x1,x2, ...
% [r,dr,intervalo]=ejemplo1_propagacion_errores(f,x,dx)
% Variables de entrada
% f cadena de caracteres que contiene la expresión de la función
% x vector que contiene las mediciones hechas
```

```
% dx vector que contiene los errores absolutos en la medida de x
% Variables de salida
% r valor predicho de la función f
% dr valor absoluto esperable en la función f
% intervalo zona de posible variación del valor exacto de f
% Ejemplo
%[r,dr,intervalo]=propagacion_directa_errores('sqrt(x1^2+x2^2)',[16,20
],[0.01,0.001])

syms x1 x2 x3 x4 x5 x6 x7 x8 x9 x10

% leemos cuántas variables tenemos
n=length(x);

% creamos el array de variables a sustituir por valores
vari='{x1';
for i=2:n
    vari=[vari,',x',num2str(i)];
end
vari=[vari,'}'];

% valor de f
r=double(subs(f,vari,x));

% estimación del valor absoluto en el cálculo de r
dr=abs(double(subs(diff(f,x1),vari,x)))*dx(1);

for i=2:n
    vari_i=['x',num2str(i)];
    dr=dr+abs(double(subs(diff(f,vari_i),vari,x)))*dx(i);
end
dr
% intervalo
intervalo=[r-dr,r+dr]
```

Propagación inversa de errores.

```
function [dx]=propagacion_inversa_errores(f,x,tol)

% Esta función resuelve de manera aproximada el problema inverso
% de la propagación de errores con una función de n (n<=10) variables
f(x1,x2,...,xn)
% [dx]=propagacion_inversa_errores(f,x,tol)
% Variables de entrada
% f cadena de caracteres que contiene la función que vamos a calcular
% x aproximación a las mediciones esperadas
% tol error absoluto máximo permitido en la función
% Variables de salida
% dx valor absoluto permitido en la medición de cada una de las
variables

syms x1 x2 x3 x4 x5 x6 x7 x8 x9 x10

% leemos cuántas variables tenemos
n=length(x);

% creamos el array de variables a sustituir por valores
```

```
vari='{x1';
for i=2:n
    vari=[vari,',x',num2str(i)];
end
vari=[vari,'}'];


% suma de las derivadas parciales evaluadas en el punto pedido
suma=abs(double(subs(diff(f,x1),vari,x)));

for i=2:n
    vari_i=['x',num2str(i)];
    suma=suma+abs(double(subs(diff(f,vari_i),vari,x)));
end


% valor absoluto permitido en la medición de cada una de las variables

dx=tol/suma;
```

Método de Horner.

```
function [d,coe]=metodo_de_Horner(p,c,m)

% Este programa evalúa un polinomio y sus derivadas en un punto c
% con menos operaciones que la evaluación directa
% [d,coe]=metodo_de_Horner(p,c,m)
% Variables de entrada:
% p es un vector que contiene los coeficientes del polinomio
% c es el valor donde queremos evaluar el polinomio
% m es el número de derivadas que queremos calcular
%Variables de salida
% d polinomio con los resultados para cada derivada

%Ejemplo
%[d,coe]=metodo_de_Horner([1 -6 8 8 4 -4],3,3)

% Grado del polinomio
n=length(p);

% definimos d como una dato tipo celda que contiene los diferentes
pasos
% al realizar Horner
coe={p};


factorial=1;

for i=1:m+1

b(1)=p(1);
for k=2:n
    b(k)=p(k)+c*b(k-1);
end

d(i)=b(n)*factorial;
```

```
coe{i+1}=b(1:n-1);

% actualización en el bucle
factorial=factorial*i;
p=b(1:n-1);
clear b;
n=n-1;

end
```

Ecuación Segundo grado.

```
function
[usual,recomendado,error_usual,error_recomendado]=ecuacion_segundo_gra
do(p,raices)

% Esta función resuelve la ecuación de segundo grado por la fórmula
usual,
% y también por el algoritmo recomendado que evita la cancelación
% catastrófica que se da al restar dos terminos muy parecidos
% Esta planteado para calcular los errores respecto de las raíces
exactas
% de la ecuación que se suponen conocidas o que se calculan
previamente con
% aritmética exacta
%
[usual,recomendado,error_usual,error_recomendado]=ecuacion_segundo_gra
do(p,raices);
% Variables de entrada:
% p vector que contiene los coeficientes del polinomio. Por ejemplo
%          en p(x)=x^2+5x+6, el vector p=[1 5 6]
% raices vector que contiene las dos raíces exactas del polinomio p
% Variables de salida:
% usual vector que contiene las dos aproximaciones a las raíces
calculadas
%       por la fórmula usual (-b+sqrt(b^2-4ac))/(2a), (-b-sqrt(b^2-
4ac))/(2a)
% recomendado vector que contiene las dos aproximaciones a las raíces
%       calculadas usando en algunos casos la fórmula
%       2c/(-b+sqrt(b^2-4ac)),  2c/(-b-sqrt(b^2-4ac))
% error_usual error relativo cometido entre las raíces exactas y las
calculadas por
%       el método usual
% error_recomendado error relativo cometido entre las raíces exactas y
las
%      calculadas por el método recomendado
% Ejemplo:
% x^2-(10^5+10^(-5))x+1=0
%
[usual,recomendado,error_usual,error_recomendado]=ecuacion_segundo_gra
do([1 -(10^5+10^(-5)) 1],[10^5 10^(-5)])



% asignamos los coeficientes
a=p(1); b=p(2); c=p(3);

% Calculamos las raíces con el método usual
```

```
usual(1)=(-b+sqrt(b^2-4*a*c))/(2*a);

usual(2)=(-b-sqrt(b^2-4*a*c))/(2*a);

% Calculamos las raíces con el método recomendado

if b>=0
    recomendado(1)=2*c/(-b-sqrt(b^2-4*a*c));
    recomendado(2)=(-b-sqrt(b^2-4*a*c))/(2*a);
else
    recomendado(1)=(-b+sqrt(b^2-4*a*c))/(2*a);
    recomendado(2)=2*c/(-b+sqrt(b^2-4*a*c));
end


% Calculamos los errores cometidos al aproximar las raíces

error_usual=abs(raices-usual)./raices;
error_recomendado=abs(raices-recomendado)./raices;
```

Ejemplo de Wilkinson.

```
function [raices_aprox,errores]=ejemplo_Wilkinson(coef,epsilon)


% Esta función desarrolla el famoso ejemplo de Wilkinson, en el cual
% se construye el polinomio de raíces 1,2,...,20, y posteriormente se
% calculan sus raíces con un método de cálculo de raíces para
polinomios
% despúes de alterar levemente el coeficiente de x^(19) por un epsilon
% dado, por ejemplo epsilon=10^(-7). En el programa se permite alterar
% otro coeficiente y con otros valores de epsilon obteniendo
resultados
% semejantes.
% [raices_aprox,errores]=ejemplo_Wilkinson(coef,epsilon);
% Variables de entrada:
% coef coeficiente que va a ser alterado, contado desde el director
% epsilon variación que se le añade al coeficiente alterado
% Variables de salida:
% Ejemplo:
% [raices_aproximadas,errores]=ejemplo_Wilkinson(2,10^(-7));


% Formamos el polinomio que tiene por raíces 1,2,...,20

r=1:20;
r=r';
p=poly(r);

% Alteramos el coeficiente determinado sumándole epsilon

p(coef)=p(coef)+epsilon;

% Calculamos las raíces del nuevo polinomio
```

```
raices_aprox=roots(p);

% Calculamos los errores cometidos al aproximar las raíces

errores=r(20:-1:1)-raices_aprox;
```

Capitulo II.

Método de bisección.

```
function [root,err,n]=bisect(func,a,b,varargin)

% Esta función aproxima las raices de una función en el intervalo
mediante
% el método de bisección.
% [root,err]=bisect(func,a,b,tol,tolf)
% Variables de entrada:
% func es la función de la que queremos encontrar sus raices
introducida
% como cadena
% a y b son el extremo izquierdo y el extremo derecho del intervalo.
% tol=tolerancia del error en x ( por defeto es 1.0e4*eps)
% tolf= tolerancia de aproximación de la función a cero. Por defecto
eps
% Variables de salida
% root=raiz aproximada
% err=error cometido en la última iteración.
% n=número de iteraciones realizadas.
% Ejemplo:
% [root,err,n]=bisect('exp(x)-3*x',0,1)

if nargin==4
    tol=varargin{1};
    tolf=varargin{2};
elseif nargin==3
    tol=varargin{1};
    tolf=1.0e6*eps;
elseif nargin<3
    tol=1.0e6*eps;
    tolf=1.0e6*eps;
end

%Condiciones de convergencia.
%1.f(a)*f(b)<0

fa=subs(func,a);
fb=subs(func,b);

if abs(fa)<tolf
    root=a;
    err=0;
    n=0;
    return;
end
```

```
if abs(fb)<tolf
    root=b;
    err=0;
    n=0;
    return;
end

if fa*fb>0
    error ('La raiz no se encuentra en el intervalo(a,b)')
end
```

Teorema del punto fijo.

```
function [j,p,err,P]=fixpt(g,p0,varargin)

% Esta función realiza la aproximación de raices de una función en un
% intervalo[a,b] por el método iterativo general o teorema del punto
fijo.
% [k,p,err,P]=fixpt(g,p0,a,b,max1,tol)
% Variables de entrada
% g es la función de iteración que tiene que introducirse como una
cadena
%   de caracteres
% p0 es el punto de partida
% max1 es el número máximo de iteraciones
% tol es la tolerancia utilizada como criterio de parada. Si dos
iterados
%     sucesivos distan menos que tol el programa para
% Variables de salida
% j es el número de iteraciones realizadas
% p es la aproximación al punto fijo
% err es la diferencia entre dos términos consecutivos
% P es la sucesión {pn} completa
% Ejemplo
% [j,p,err,P]=fixpt('sqrt(exp(x)/3)',1,0,1)

P(1)=p0;

if nargin==4
    max1=varargin{1};
    tol=varargin{2};
elseif nargin==3
    if varargin{1}<=1/1000
        tol=varargin{1};
        max1=100;
    else
    max1=varargin{1};
    tol=1.0e6*eps;
    end
elseif nargin<3
    tol=1.0e6*eps;
    max1=100;
end
```

```
%Comenzamos las iteraciones.

for j=2:max1
    P(j)=subs(g,P(j-1));
    err=abs(P(j)-P(j-1));
    relerr=err/(abs(P(j))+eps);
    p=P(j);
    if (err<tol)||(relerr<tol)
        break
    end
end
```

Método de Lagrange o regula falsi.

```
function [c,err,yc]=regula(f,a,b,varargin)

% Esta función realiza la aproximación de funciones en un intervalo
[a,b]
% por el método de Lagrange o regula falsi.
% [c,err,yc]=regula(f,a,b,max1,delta,epsilon)
% Variables de entrada.
% f es una función, introducida como una cadena de caracteres.
% a y b son los extremos del intervalo
% max1 es el número maximo de iteraciones
% delta es la tolerancia para el criterio de parada en x
% epsilon es la tolerancia para el criterio de parada en el valor de f
% Variables de salida
% c es la raíz aproximada
% err es el error estimado de la aproximación
% yc=f(c)
%
% Ejemplo
% [c,err,yc]=regula('exp(x)-3*(x^2)',3,4)

epsilon=1.0e6*eps;
if nargin==4
    max1=varargin{1};
    delta=varargin{2};
elseif nargin==3
    if varargin{1}<=1/1000
        delta=varargin{1};
        max1=100;
    else
    max1=varargin{1};
    delta=1.0e6*eps;
    end
elseif nargin<3
    delta=1.0e6*eps;
    max1=100;
end


%Comenzamos las iteraciones

for k=1:max1
    dx=yb*(b-a)/(yb-ya);
    c=b-dx;
```

```
    ac=c-a;
    yc=subs(f,c);
    if yc<eps
        break
    elseif yb*yc>0
        b=c;
        yb=yc;
    else
        a=c;
        ya=yc;
    end

    dx=min(abs(dx),ac);
    if abs(dx)<delta
        break
    end
    if abs(yc)<epsilon
        break
    end
end

yc=subs(f,c);
err=dx;
```

Método de Newton-Raphson.

```
function [root,err,i]=newtonRaphson(func,a,b,x,varargin)


% Esta función aproxima la raíz de una fución en un intervalo [a,b]
% mediante el método de Newton Raphson.
% [root,err]=newtonRaphson(func,a,b,x,n,tol)
% Variables de entrada:
% func función de la que queremos aproximar la raíz introducida como
%      cadena de caracteres.
% a y b son el extremo izquierdo y el extremo derecho del intervalo
% x punto inicial para comenzar las iteraciones
% tol es la tolerancia admitida para considararlo como raiz. Por
defecto
%     tol=1.0e6*eps.
% n es el número de iteraciones a realizar. Por defecto n=10
% Variables de salida:
% root es la raiz aproximada.
% err es la distancia entre los dos últimos iterados
%
% Ejemplo.
%[root,err]=newtonRaphson('exp(x)-3*(x^2)',-1,0,-0.5)

if nargin==6
    n=varargin{1};
    tol=varargin{2};
elseif nargin==5
    if varargin{1}<=1/1000
        tol=varargin{1};
        n=100;
    else
    n=varargin{1};
    tol=1.0e6*eps;
```

```
    end
elseif nargin<5
    tol=1.0e6*eps;
    n=100;
end


% Comprobación de existencia de raíz
%1.f(a)*f(b)<0
fa=subs(func,a);
fb=subs(func,b);
if abs(fa)<eps
    root=a;
    err=0;
    return;
end
if abs(fb)<eps
    root=b;
    err=0;
    return;
end
if fa*fb>0
    error ('La raíz no se encuentra en el intervalo [a,b]')
end


% Comenzamos el proceso de iteraciones.

for i=1:n
    fx=subs(func,x);
    if abs(fx)<eps
        root =x;
        err=0;
        return;
    end

    % Paso de Newton-Raphson
    dfx=subs(diff(func),x);
    if abs(dfx)==0
        error('La derivada es cero y no puede haber convergencia');
    else
        dx=-fx/dfx;
    end
    x=x+dx;

    % Comprueba la convergencia
    if abs(dx)<tol
        root=x;
        err=abs(dx);
        return
    end
end

root=x;
err=abs(dx);
```

Método de la secante.

```
function [root,err,k,y]=secant(f,p0,p1,varargin)

% Esta función realiza la aproximación de la raíz de una función dadas
unas
% aproximaciones iniciales de la raiz p0 y p1
% [p1,err,k,y]=secant(f,p0,p1,max1,delta,epsilon)
% Variables de entrada
% f es la función pasada como cadena de caracteres
% p0 y p1 son las aproximaciones iniciales a un cero de f
% max1 es el número máximo de iteraciones
% delta es la tolerancia en x
% epsilon es la tolerancia para los valores de la función
% Variables de salida.
% root es la aproximación al cero
% err es una estimación del error en la aproximación
% k es el número de iteraciones realizadas
% y es el valor de la función en la raíz aproximada f(root)
% Ejemplo.
% [root,err,k,y]=secant('cos(x)-x',0.5,pi/4)


epsilon=1.0e6*eps;

if nargin==5
    max1=varargin{1};
    delta=varargin{2};
elseif nargin==4
    if varargin{1}<=1/10
        delta=varargin{1};
        max1=100;
    else
    max1=varargin{1};
    delta=1.0e6*eps;
    end
elseif nargin<4
    delta=1.0e6*eps;
    max1=100;
end


for k=1:max1
    p2=p1-subs(f,p1)*(p1-p0)/(subs(f,p1)-subs(f,p0));
    err=abs(p2-p1);
    relerr=err/(abs(p2)+eps);
    p0=p1;
    p1=p2;
    y=subs(f,p1);
    if (err<delta)|(relerr<delta)|(abs(y)<epsilon)
        break
    end
end
root=p1;
```

Método de la convergencia de Aitken.

```
function [xg,i,k,yg]=Delta2_Aitken(x0,g,varargin)

% Este programa ejecuta el metodo de aceleración Delta2 de Aitken
% [xg,i,k,yg]=Delta2_Aitken(x0,g,tol,nmax)
% Variables de entrada:
% x0 aproximacion inicial a la raíz
% g  función del metodo de punto fijo de orden lineal
% nmax número máximo de iteraciones permitido
% tol tolerancia para el criterio de parada. Si dos iterados sucesivos
%     distan menos que tol el programa para
% Variables de salida:
% xg aproximación a la raíz
% i número de aceleraciones realizado
% k número total de iteraciones realizadas
% yf valor de la funcion en xf
% Ejemplo:
% Para resolver la ecuacion x^2-x=0
% Para hallar el punto x=1 en la ecuacion x=sqrt(x)
% [xg,i,k,yg]=Delta2_Aitken(0.5,'sqrt(x)',1000,10^(-6)


if nargin==4
    nmax=varargin{1};
    tol=varargin{2};
elseif nargin==3
    if varargin{1}<=1/1000
        tol=varargin{1};
        nmax=100;
    else
    nmax=varargin{1};
    tol=1.0e6*eps;
    end
elseif nargin<3
    tol=1.0e6*eps;
    nmax=100;
end

for i=1:nmax
    x1=subs(g,x0);
    if abs(x1-x0)<tol
        xg=x1; yg=subs(g,x1);
        k=3*(i-1)+1;
        return;
    end
    x2=subs(g,x1);
    if abs(x2-x1)<tol
        xg=x2; yg=subs(g,x2);
        k=3*(i-1)+2;
        return;
    end
    x0=x0-((x1-x0)^2)/(x2-2*x1+x0);
    if abs(x2-x0)<tol
        xg=x0; yg=subs(g,x0);
        k=3*(i-1)+3;
        return;
    end
end
```

```
xg=x0; yg=subs(g,xg);
k=3*i;
```

Acotación de raíces de una ecuación polinómica.

```
function [L l]=acotacion_de_polinomios(p,Lmax)


% Esta función permite realizar la acotación de raices de un polinomio
% mediante el método de Laguerre-Thibault.
% [L l]=acotacion_de_polinomios(p,Lmax)
% Variables de entrada:
% p es el vector que contiene los coeficientes del polinomio
% Lmax es un valor máximo inicial de la posible cota superior del
polinomio
%      (-Lmax posible cota inferior)
% Variables de salida.
% L es la cota superior de las raices.
% l es la cota inferior de las raices.
% Ejemplo.
% [L l]=acotacion_de_polinomios([1 -2 +1 -1 +2 -3 +7 -1],10)

n=length(p);

% Necesitamos que el coeficiente director sea positivo
if p(1)<0
    p=-p;
end

% Probamos por los números enteros sucesivamente

for i=1:Lmax

 b(1)=p(1);
 k=1;
 while b(k)>=0 && k<n
    k=k+1;
    b(k)=p(k)+i*b(k-1);
 end

if k==n && b(n)>=0
    L=i;
    break
end

end

if isempty(L)==1
    disp('La cota Lmax es insuficiente para la acotación superior');
end

% Ahora calculamos la cota inferior mediante un cambio de variable

q=p;
```

```
if mod(n,2)==0
    q(1:2:n-1)=-p(1:2:n-1);
else
    q(2:2:n-1)=-p(2:2:n-1);
end


% Necesitamos que el coeficiente director sea positivo
if q(1)<0
    q=-q;
end

% Probamos por los números enteros sucesivamente

for i=1:Lmax

 b(1)=q(1);
 k=1;
 while b(k)>=0 && k<n
    k=k+1;
    b(k)=q(k)+i*b(k-1);
 end

if k==n && b(n)>=0
    l=-i;
    break
end

end

if isempty(l)==1
    disp('La cota Lmax es insuficiente para la acotación inferior');
end
```

*Jacobiano

```
function J=jacobiano(f,v)

% Esta función devuelve la matriz jacobiana de una función de varias
% variables f
% Las variables que intervienen deberán haber
% sido declaradas como variables simbólicas con anterioridad.
% J=jacobiano(f,v)
% Variables de entrada:
% f matriz que contiene el sistema de ecuaciones
% v vector que contiene las variables que intervienen en la función f
% Variables de salida:
% J matriz jacobiana del sistema de ecuaciones
%
% Ejemplo
% J=jacobiano([3*x+y+z;2*z+y^4],[x y z])

n=length(f);
k=length(v);

for i=1:n
```

```
    for j=1:k
        J(i,j)=diff(f(i),v(j));
    end
end
```

Método de Newton para sistemas no lineales.

```
function [P,iter,err]=newdim(f,v,P0,varargin)


% Resolución del sistema de ecuaciones no lineales F(X)=0, generando,
a
% partir de una aproximación inicial P0, una sucesión{Pk} que converge
a la
% solución P
% [P,iter,err]=newdim(f,v,P0,max1,delta,epsilon)
% Variables de entrada:
% f es el sistema no lineal.
% v es el vector con las variables que intervienen en el sistema.
% P0 es el punto inicial
% max1 es el numero máximo de iteraciones
% delta es la tolerancia para P.
% epsilon es la tolerancia para F(P)
% Variables de salida:
% P es la aproximación a la solución que se obtiene
% iter es el número de iteraciones realizadas
% err es la estimación del error en P

% Transformamos las variables en simbólicas
x=sym(v);

epsilon=1.0e6*eps;

if nargin==5
    max1=varargin{1};
    delta=varargin{2};
elseif nargin==4
    if varargin{1}<=1/1000
        delta=varargin{1};
        max1=100;
    else
    max1=varargin{1};
    delta=1.0e6*eps;
    end
elseif nargin<4
    delta=1.0e6*eps;
    max1=100;
end



Y=-subs(f,x,P0)';
P=P0';
JF=jacobiano(f,x);
for iter=1:max1
    J=subs(JF,x,P);
    Dx=J\Y;
```

```
    Q=P+Dx;

    err=norm(Q-P);
    relerr=err/(norm(Q)+eps);
    P=Q;
    Y=-subs(f,x,Q)';

    if (err<delta)|(relerr<delta)|(abs(Y)<epsilon)
        break
    end

end
```

Capitulo III.

Sistema triangular superior.

```
function X=backsub(A,B)

% Resolución de un sistema triangular superior AX=B por el método de
% sustitución regresiva. El método funciona sólo si todos los
elementos
% diagonales son distintos de cero.
% X=backsub(A,B);
% Variables de entrada
% A es una matriz triangular superior invertible de orden nxn.
% B es una matriz de orden nx1.
% Variables de salida
% X es la solución del sistema lineal AX=B.
% Ejemplo.
% X=backsub([3 -2 1 -1;0 4 -1 2;0 0 2 3;0 0 0 5],[8 -3 11 15])

% Cálculo de la dimensión de B e iniciación de X.

n=length(B);
X=zeros(n,1);

% Resolución del sistema por sustitución regresiva

X(n)=B(n)/A(n,n);
for k=n-1:-1:1
    X(k)=(B(k)-sum(A(k,k+1:n).*X(k+1:n)'))/A(k,k);
end
```

Método de Gauss.

```
function x =gauss(A,b)

% Esta función resuelve el sistema lineal AX=B mediante el método de
Gauss
% x=gauss(A,b)
% Variables de entrada:
% A matriz de coeficientes del sistema lineal
```

```
% b vector de terminos independientes del sistema lineal
% Variables de salida:
% x vector solución del sistema Ax=b.
% Ejemplo:
% x =gauss([2 4 1;2 6 -1;1 5 2],[4 10 2]);



if size(b,2)>1
    b=b';
end

n=length(b);

% Bucle para realizar los diferentes pasos de Gauss


for k= 1:n-1

    if A(k,k)<eps
         uiwait(msgbox('La matriz debe ser invertible', 'Mensaje de
error',...
                       'error','modal'))
                   return;
    end

% Hace un paso de eliminación de Gauss

for i=k+1:n
    if A(i,k)~=0
        lambda=A(i,k)/A(k,k);
        A(i,k+1:n)=A(i,k+1:n)-lambda*A(k,k+1:n);
        b(i)=b(i)-lambda*b(k);
    end
end

end



% Resolución por sustitución regresiva

for k=n:-1:1
    b(k)=(b(k)-A(k,k+1:n)*b(k+1:n))/A(k,k);
end
x=b;
```

Método de Gauss con pivoteo parcial.

```
function x =gaussPiv_Parcial(A,b)

% Esta función resuelve el sistema lineal AX=B mediante el método de
Gauss
% con pivoteo parcial seleccionando previamente el pivote más
conveniente
% a cada paso de Gauss dentro de su misma columna
```

```
% x =gaussPiv_Parcial(A,b)
% Variables de entrada:
% A matriz de coeficientes del sistema lineal
% b vector de terminos independientes del sistema lineal
% Variables de salida:
% x vector solución del sistema Ax=b.
% Ejemplo:
% x =gaussPiv_Parcial([1 2 1 4;2 0 4 3;4 2 2 1;-3 1 3 2],[13 28 20 6])



if size(b,2)>1
    b=b';
end

n=length(b);
s=zeros(n,1);

% Establecemos el factor de escala para cada fila

for i=1:n
    s(i)=max(abs(A(i,1:n)));
end

% Bucle para realizar los diferentes pasos de Gauss

for k= 1:n-1

    % Buscamos el pivote más apropiado en la misma columna
    % dividiendo previamente los elementos por su factor de escala

    [Amax,p]=max(abs(A(k:n,k))./s(k:n));
    p=p+k-1;
    if Amax<eps
        error ('La matriz es singular')
    end

    % Intercambiamos filas si es necesario

    if p~=k
        b=swapRows(b,k,p);
        s=swapRows(s,k,p);
        A=swapRows(A,k,p);
    end

% Hace un paso de eliminación de Gauss

for i=k+1:n
    if A(i,k)~=0
        lambda=A(i,k)/A(k,k);
        A(i,k+1:n)=A(i,k+1:n)-lambda*A(k,k+1:n);
        b(i)=b(i)-lambda*b(k);
    end
end

end
```

```
% Resolución por sustitución regresiva

for k=n:-1:1
    b(k)=(b(k)-A(k,k+1:n)*b(k+1:n))/A(k,k);
end
x=b;
```

Método de Gauss con pivoteo total.

```
function x =gaussPiv_Total(A,b)

% Esta función resuelve el sistema lineal AX=B mediante el método de
Gauss
% con pivoteo parcial seleccionando previamente el pivote más
conveniente
% a cada paso de Gauss dentro de su misma columna
% x =gaussPiv(A,b)
% Variables de entrada:
% A matriz de coeficientes del sistema lineal
% b vector de terminos independientes del sistema lineal
% Variables de salida:
% x vector solución del sistema Ax=b.
% Ejemplo:
% x =gaussPiv_Total([1 2 1 4;2 0 4 3;4 2 2 1;-3 1 3 2],[13 28 20 6])



if size(b,2)>1
    b=b';
end

n=length(b);

% inicializamos el vector de incógnitas

incog=1:n;


% Bucle para realizar los diferentes pasos de Gauss

for k= 1:n-1

    % Buscamos el pivote más apropiado en toda la submatriz de orden
    % n-k+1


    [maxC,F]=max(abs(A(k:n,k:n)),[],1);
    [Amax,pcolumna]=max(maxC);
    pfila=F(pcolumna);


    pfila=pfila+k-1;
    pcolumna=pcolumna+k-1;

    if Amax<eps
         error ('La matriz es singular')
```

```
    end

    % Intercambiamos filas y columnas si es necesario

    if pfila~=k
        b=swapRows(b,k,pfila);
        A=swapRows(A,k,pfila);
    end
    if pcolumna~=k
        A=swapColumns(A,k,pcolumna);
        aux=incog(k);
        incog(k)=incog(pcolumna);
        incog(pcolumna)=aux;
    end

    % Hace un paso de eliminación de Gauss

    for i=k+1:n
        if A(i,k)~=0
            lambda=A(i,k)/A(k,k);
            A(i,k+1:n)=A(i,k+1:n)-lambda*A(k,k+1:n);
            b(i)=b(i)-lambda*b(k);
        end
    end


end


% Resolución por sustitución regresiva

for k=n:-1:1
    b(k)=(b(k)-A(k,k+1:n)*b(k+1:n))/A(k,k);
end
x=b;
```

Factorización LU.

```
function [X,LU,P]=lufact(A,B)

% PA=LU:factorización con pivoteo. Cálculo de la solución de un
sistema AX=B
% cuando la matriz A es invertible.
% X=lufact(A,B)
% Variables de entrada.
% A es una matriz de orden NxN.
% B es una metriz de orden Nx1.
% Variables de salida.
% X es la matriz de orden Nx1 solución de AX=B.
% LU contiene la factorización LU. L por debajo de la diagonal y U por
% arriba.
% P contiene las permutaciones de fila realizadas
% Ejemplo.
% X=lufact([1 4 -2;3 -2 5;2 3 1],[3 14 11])
```

```
%Iniciamos X,Y, la matriz de almacenamiento temporal C y la matriz
fila R
%donde se registran los intercambios de filas.
N=length(b);
X=zeros(N,1);
Y=zeros(N,1);
C=zeros(1,N);
P=1:N;
for q=1:N-1
    %Determinación de la fila pivote para la columna q-ésima.
    [max1,j]=max(abs(A(q:N,q)));
    %Intercambio de las filas q-ésima y j-ésima.
    C=A(q,:);
    A(q,:)=A(j+q-1,:);
    A(j+q-1,:)=C;
    d=R(q);
    P(q)=P(j+q-1);
    P(j+q-1)=d;
    if A(q,q)==0
        disp('A es singular,no hay solución o no es única');
        break
    end
    %Cálculo del multiplicador que se guarda en la parte subdiagonal
de A.
    for k=q+1:N
        mult=A(k,q)/A(q,q);
        A(k,q)=mult;
        A(k,q+1:N)=A(k,q+1:N)-mult*A(q,q+1:N);
    end
end
LU=A;
%Resolución para hallar Y.
Y(1)=B(R(1));
for k=2:N
    Y(k)=B(R(k))-A(k,1:k-1)*Y(1:k-1);
end
%Resolución para hallar X.
X(N)=Y(N)/A(N,N);
for k=N-1:-1:1
    X(k)=(Y(k)-A(k,k+1:N)*X(k+1:N))/A(k,k);
end
```

Factorización de Choleski.

```
function [x,L] =choleskiSol(A,b)

% Esta función resielve el sistema de ecuaciones lineal AX=B por el
método
% de cholesky, donde la matriz L ha sido previamente calculada a
partir de
% la factorización, según Cholesky, de A.
% [x,L] =choleskiSol(L,b);
% Variables de entrada.
% b es el vector de términos independientes del sistema lineal de
ecaciones.
% Variables de salida.
% x es el vector solución del sistema lineal Ax=b.
% L factorización de Cholesky.
% Ejemplo
```

```
% [x,L] =choleskiSol([1 -1 1;-1 5 1;1 1 3],[1 1 0])

n=size(A,1);
L=zeros(n);
for j= 1:n
    temp = A(j,j)-dot(L(j,1:j-1),L(j,1:j-1));
    if temp <0.0
           error('La matriz no es definida positiva')
    end
    L(j,j)=sqrt(temp);
    for i= j+1:n
        L(i,j)=(A(i,j)-dot(L(i,1:j-1),L(j,1:j-1)))/L(j,j);
    end
end


%Resuelve LL'*x=b
n=length(b);
if size(b,2)>1;
    b=b';
end
%Solución Ly=b
for k=1:n
    b(k)=(b(k)-dot(L(k,1:k-1),b(1:k-1)'))/L(k,k);
end
%Solución L'x=y
for k= n:-1:1
    b(k)=(b(k)-dot(L(k+1:n,k),b(k+1:n)))/L(k,k);
end
x=b;
```

Método iterativo de Jacobi.

```
function X=jacobi(A,B,P,varargin)

% Resolución de un sistema lineal AX=B mediante la generación de una
% sucesión {Pk} que converge a la solución, a partir de un punto
inicial
% P0.Una condición suficiente para que el método sea aplicable es que
A sea
% diagonal estrictamente dominante.
% Variables de entrada:
% A es una matriz invertible de orden NxN.
% B es una matriz de orden Nx1.
% P es una matriz de orden Nx1: el punto inicial.
% max1 es el número máximo de iteraciones.
% delta es la tolerancia para P.
% Variables de salida.
% X es una matriz de orden Nx1:la aproximación a la solución de AX=B
% generada por el método iterativo de jacobi.
% Ejemplo:
% X=jacobi([4 -1;1 5],[15 9],[0 0],1.0e6*eps,30)

if nargin==5
    max1=varargin{1};
    delta=varargin{2};
elseif nargin==4
    if varargin{1}<=1/1000
```

```
        delta=varargin{1};
        max1=100;
    else
        max1=100;
        delta=1.0e6*eps;
    end
elseif nargin<4
    max1=100;
    delta=1.0e6*eps;
end


N=length(B);
for k=1:max1
    for j=1:N
        X(j)=(B(j)-A(j,[1:j-1,j+1:N])*P([1:j-1,j+1:N]))/A(j,j);
    end
    err=abs(norm(X'-P));
    relerr=err/(norm(X)+eps);
    P=X';
    if (err<delta)||(relerr<delta)
        break
    end
end
X=X';
```

Método iterative de Gauss-Seidel.

```
function P=gseid(A,b,P,varargin)

% Resolución de un sistema lineal AX=B mediante la generación de una
% sucesión{Pk} que converge a la solución, a partir de un punto
inicial
% P0.Una condición suficiente para que el método sea aplicable es que
A sea
% de diagonal estrictamente dominante.
% X=gseid(A,B,P,delta,max1)
% Variables de entrada
% A es la matriz invertible de orden NxN
% B es la matriz de orden Nx1
% P es una matriz de ordenNx1:el punto inicial
% max1 es el número máximo de iteraciones
% delta es la tolerancia para P
% Variables de salida
% X es una matriz de orden Nx1;
% la aproximación a la solución AX=B generada por el método iterativo
de
% Gauss-Seidel.
% Ejemplo:
% P=gseid([4 -1;1 5],[15;9],[0;0],30,1.0e6*eps)

if nargin==5
    max1=varargin{1};
    delta=varargin{2};
elseif nargin==4
    if varargin{1}<=1/10
        delta=varargin{1};
        max1=100;
```

```
    else
        max1=varargin{1};
        delta=1.0e6*eps;
    end
elseif nargin<4
    max1=100;
    delta=1.0e6*eps;
end


N=length(b);
err=0;
for k=1:max1
    for j=1:N

        aux=(b(j)-A(j,1:j-1)*P(1:j-1)-A(j,j+1:N)*P(j+1:N))/A(j,j);
        err=(aux-P(j))^2;
        P(j)=aux;

    end

    err=sqrt(err);
    relerr=err/(norm(P)+eps);

    if (err<delta)||(relerr<delta)
        break
    end
end
```

Capitulo IV.

Interpolación de Lagrange.

```
function [C,L,Ye]=Lagrange(X,Y,varargin)

% Construcción del polinomio interpolador de Lagrange que pasa por los
N+1
% puntos(xk,yk) para k=0,1,...,N.
% [C,L,Ye]=Lagrange(X,Y,Xe)
% Variables de entrada
% X es un vector que contiene la lista de las abscisas.
% Y es un vector que contiene la lista de las ordenadas.
% Xe vector de abscisas donde se quiere evaluar el polinomio
% Variables de salida
% C es el vector que contiene los coeficientes del polinomio
interpolador
% de Lagrange.
% L es la matriz que contiene los coeficientes de los polinomios
% coeficientes de Lagrange.
% Ye evaluación del polinomio de Lagrange en las abscisas Xe
% Ejemplo.
% Interpolamos la función y=f(x)=cos(x) utilizando los nodos x0=0.0
x1=0.6
% x2=1.2, donde será y0=1 y1=0.825336 e y2=0.362358
%[C,L,Ye]=Lagrange([0 0.6 1.2],[1 0.825336 0.362358])
```

```
if nargin==3
    Xe=varargin{1};
else
    Xe='';
    Ye='';
end


w=length(X);
L=zeros(w,w);

%Formación de los polinomios coeficientes de Lagrange.

for k=1:w

    V=1;
    for j=1:w

        if k~=j
            V=conv(V,poly(X(j)))/(X(k)-X(j));
        end

    end

    L(k,:)=V;

end

% Cálculo de los coeicientes del polinomio interpolador de Lagrange.

C=Y*L;


% Evaluación en Xe

if ~isempty(Xe)
    Ye=polyval(C,Xe);
end

% pasamos el polinomio a variable simbólica

syms x
for i=0:w-1
    b(i+1)=x^(w-i-1);
end
Cx=dot(C,b);


% dibujo del polinomio de Lagrange y de los puntos iniciales

% polinomio
dx=(X(w)-X(1))/10;
A=X(1)-dx;
B=X(w)+dx;
h1=ezplot(Cx,[A,B]);
```

```
% puntos iniciales
hold on
h2=plot(X,Y,'ro','MarKerSize',10);

if ~isempty(Xe)
    h3=plot(Xe,Ye,'g*','MarKerSize',5);
    legend([h1,h2,h3],'Polinomio de Lagrange','Nodos de
interpolación','Evaluación en las Abscisas');
else
    legend([h1,h2],'Polinomio de Lagrange','Nodos de interpolación');
end
```

Método de interpolación de Newton.

```
function [C,N,Ye]=Newton(X,Y,varargin)

% Construcción del polinomio interpolador de Newton que pasa por los
N+1
% puntos(xk,yk) para k=0,1,...,N.
% [C,N,Ye]=Newton(X,Y,Xe)
% Variables de entrada
% X es un vector que contiene la lista de las abscisas.
% Y es un vector que contiene la lista de las ordenadas.
% Xe vector de abscisas donde se quiere evaluar el polinomio
% Variables de salida
% C es el vector que contiene los coeficientes del polinomio
interpolador
% de Newton.
% N es la matriz que contiene los coeficientes de los polinomios
% base para construir el polinomio de Newton.
% Ye evaluación del polinomio de Newton en las abscisas Xe
% Ejemplo.
%[C,N,Ye]=Newton([0 1 2 3 4],[1 0.5403023 -0.4161468 -0.98999925 -
0.6536436])

if nargin==3
    Xe=varargin{1};
else
    Xe='';
    Ye='';
end


w=length(X);
N=zeros(w,w);

% Cálculo de las diferencias divididas

D=diferencias_divididas(X,Y);


% Formación de los polinomios coeficientes de Newton.

for k=1:w

    V=1;
    for j=1:k-1
```

```
        V=conv(V,poly(X(j)));
    end

    N(k,w-length(V)+1:w)=V;

end

% Cálculo de los coeficientes del polinomio interpolador de Lagrange.

C=D*N;


% Evaluación en Xe

if ~isempty(Xe)
    Ye=polyval(C,Xe);
end


syms x
for i=0:w-1
    b(i+1)=x^(w-i-1);
end
Cx=dot(C,b);


% dibujo del polinomio de Newton y de los puntos iniciales

% polinomio
dx=(X(w)-X(1))/10;
A=X(1)-dx;
B=X(w)+dx;
h1=ezplot(Cx,[A,B]);

% puntos iniciales
hold on
h2=plot(X,Y,'ro','MarKerSize',10);

if ~isempty(Xe)
    h3=plot(Xe,Ye,'g*','MarKerSize',5);
    legend([h1,h2,h3],'Polinomio de Newton','Nodos de
interpolación','Evaluación en las Abscisas');
else
    legend([h1,h2],'Polinomio de Newton','Nodos de interpolación');
end
```

*Tabla de diferencias divididas.

```
function [fila1,tabla]=diferencias_divididas(X,Y)

% Este programa calcula las diferencias divididas
% [fila1,tabla]=diferencias_divididas(X,Y);
% Variables de entrada:
% X son las abscisas
% Y son las ordenadas
```

```
% Variables de salida:
% fila1 contiene los coeficientes para formar el polinomio
interpolador
%     de Newton
% tabla contiene el cuadro completo de las diferencias divididas
%
% Ejemplo:
% Construir la tabla de diferencias divididas con los datos
% x0=1; x1=2;
% f(x0)=1, f(x1)=-1;
% [fila1,tabla]=diferencias_divididas([1,2],[1,-1])


% número de nodos

n=length(X);


% inicializamos la tabla a cero

tabla=zeros(n,n+1);


% completamos las primeras dos columnas

tabla(:,1)=X';
tabla(:,2)=Y';

% completamos el resto de la tabla

for i=3:n+1   % columna
    for j=1:n-i+2   % fila


        tabla(j,i)=(tabla(j+1,i-1)-tabla(j,i-1))/(tabla(j+i-2)-
tabla(j,1));


    end
end

fila1=tabla(1,2:end);
```

Método de interpolación de Hermite.

```
function [C,Ye]=Hermite(X,Ycondiciones,Xe)

% Esta función construye el polinomio interpolador de Hermite y lo
evalua
% en ciertas abscisas Xe
% [C,Ye]=Hermite(X,Ycondiciones,Xe);
% Variables de entrada:
% X son las abscisas donde se imponen las condiciones
% Ycondiciones es una celda que contiene tantos vectores como nodos.
%      Cada vector contiene los datos [f(xi),f'(xi),f''(xi)/2!,
%      ...,f^(ni)(xi)/(ni)!]
```

```
% Xe valores de las abscisas donde se quiere evaluar el polinomio
% Variables de salida:
% C coeficientes del polinomio de Hermite
% Ye valores del polinomio en las abscisas Xe
%
% Ejemplo:
% Polinomio que satisface las condiciones
% x0=1; x1=2;
% p(x0)=1, p'(x0)=2
% p(x1)=-1, p'(x1)=3, p''(x1)=4
% y su evaluación en x=1.5
% [C,Ye]=Hermite([1,2],{[1,2],[-1,3,2]},1.5)


% tabla de diferencias divididas generalizadas

[D,tabla]=diferencias_divididas_generalizadas(X,Ycondiciones);

n=length(X);
Nodos=tabla(:,1);
w=length(Nodos);
N=zeros(w,w);

% Formación de los polinomios coeficientes de Hermite

for k=1:w

    V=1;
    for j=1:k-1
            V=conv(V,poly(Nodos(j)));
    end

    N(k,w-length(V)+1:w)=V;

end


 % Cálculo de los coeficientes del polinomio interpolador de Lagrange.

 C=D*N;


% Evaluación en Xe

 if ~isempty(Xe)
     Ye=polyval(C,Xe);
 end


 % pasamos el polinomio a variable simbólica

syms x
for i=0:w-1
    b(i+1)=x^(w-i-1);
end

Cx=dot(C,b);
```

```
% dibujo del polinomio de Hermite y de los puntos iniciales

% polinomio
dx=(X(n)-X(1))/10;
A=X(1)-dx;
B=X(n)+dx;
h1=ezplot(Cx,[A,B]);

% puntos iniciales
hold on

for i=1:length(Ycondiciones)
    Y(i)=Ycondiciones{i}(1);
end

h2=plot(X,Y,'ro','MarKerSize',10);

if ~isempty(Xe)
    h3=plot(Xe,Ye,'g*','MarKerSize',5);
    legend([h1,h2,h3],'Polinomio de Hermite','Nodos de
interpolación','Evaluación en las Abscisas');
else
    legend([h1,h2],'Polinomio de Hermite','Nodos de interpolación');
end
```

*Tabla de diferencias divididas generalizadas.

```
function
[fila1,tabla,nc]=diferencias_divididas_generalizadas(nodos,condiciones
)

% Este programa calcula las diferencias divididas generalizadas
%
[fila1,tabla,nc]=diferencias_divididas_generalizadas(nodos,condiciones
);
% Variables de entrada:
% nodos son las abscisas donde se imponen las condiciones
% condiciones es una celda que contiene tantos vectores como nodos.
Cada
%      vector contiene los datos [f(xi),f'(xi),f''(xi)/2!,
%      ...,f^(ni)(xi)/(ni)!]
% Variables de salida:
% fila1 contiene los coeficientes para formar el polinomio
interpolador
%     de Hermite
% tabla contiene el cuadro completo de las diferencias divididas
% nc número de condiciones que hay en cada nodo
%
% Ejemplo:
% Construir la tabla de diferencias divididas generalizadas con los
datos
% x0=1; x1=2;
% f(x0)=1, f'(x0)=2
```

```
% f(x1)=-1, f'(x1)=3, f''(x1)=4
% [fila1,tabla,nc]=diferencias_divididas_generalizadas([1,2],{[1,2],[-
1,3,2]})


% número de nodos

n=length(nodos);

% número de condiciones en cada nodo

nc(1)=length(condiciones{1});
nc_total(1)=nc(1);
for i=2:n
    nc(i)=length(condiciones{i});
    nc_total(i)=nc_total(i-1)+nc(i);
end


% inicializamos la tabla a cero

tabla=zeros(nc_total(n),nc_total(n)+1);


% completamos las primeras dos columnas

tabla(1:nc(1),1)=nodos(1)*ones(nc(1),1);
tabla(1:nc(1),2)=condiciones{1}(1)*ones(nc(1),1);

for i=2:n
    tabla(nc_total(i-1)+1:nc_total(i),1)=nodos(i)*ones(nc(i),1);
    tabla(nc_total(i-
1)+1:nc_total(i),2)=condiciones{i}(1)*ones(nc(i),1);
end


% completamos el resto de la tabla

for i=3:nc_total(n)+1   % columna
    for j=1:nc_total(n)-i+2   % fila

        % comprobamos si todos los nodos son iguales o no para aplicar
        % una regla de cálculo u otra

        if tabla(j,1)==tabla(j+i-2)  % son todos los nodos iguales
            tabla(j,i)=condiciones{find(tabla(j,1)==nodos)}(i-1);
        else
            tabla(j,i)=(tabla(j+1,i-1)-tabla(j,i-1))/(tabla(j+i-2)-
tabla(j,1));

        end

    end
end

fila1=tabla(1,2:end);
```

Método de interpolación de Taylor.

```
function[C,Ye]=polytaylor(x0,Y,a,b,varargin)

% Construcción del polinomio interpolaor de Taylor de una función f
% cerca de un punto x0, que pertenece al intervalo [a b].
% [C,Ye]=polytaylor(x0,f,n,varargin)
% Variables de entrada
% x0 es el punto cerca del cual queremos aproximar la función.
% Y es el vector que contiene los valores de las derivadas sucesivas
de la
% funció en el punto x0.
% a extremo izquierdo del intervalo de aproximación.
% b extremo derecho del intervalo de aproxmación.
% xe es el punto en el que evaluamos el polinomio
% Variables de salida
% C es el poliniomio interpolador de la función
% Ye es el valor de polinomio en el punto xe
% Ejemplo.
% f=cos(x)
% Sabiendo que f(0.5)=0.8776; f'(0.5)=-0.4794; f''(0.5)=-0.8776
% y f'''(0.5)=0.4794
% [C,Ye]=polytaylor(0.5,[0.8776 -0.4794 -0.8776 0.4794],0,2,1.5)



if nargin==6
    xe=varargin{1};
else
    xe=0;
end
n=length(Y);
fac=1;
syms x
C=0;
for k=1:n
    fac=fac*k;
    t=((x-x0)^(k))/fac;
    C=C+Y(k)*t;
end
if xe~=0
    Ye=subs(C,xe);
else
    Ye='';
end
C=char(C);

% dibujo del polinomio de Taylor y de los puntos iniciales

% polinomio
dx=(b-a)/10;
A=a-dx;
B=b+dx;
h1=ezplot(C,[A,B]);

% puntos de resolución.

if xe~=0
```

```
    h3=plot(xe,Ye,'g*','MarKerSize',5);
    legend([h1,h3],'Polinomio de Taylor','Nodos de
interpolación','Evaluación en las Abscisas');
else
    legend([h1],'Polinomio de Taylor','Nodos de interpolación');
end
```

Splines cúbicos.

```
function [ye,ind]=SplinesCubicos(X,Y,z1,zn,xe)

% Esta función evalua en un punto x los splines cúbicos
% y=SplinesCubicos(X,Y,z1,zn,xe);
% Variables de entrada:
% X abscisas de la tabla de valores
% Y ordenadas de la tabla de valores
% z1 valor de la segunda derivada de la función Spline en t1
% zn valor de la segunda derivada de la función Spline en tn
% xe punto donde queremos evaluar la función polinómica a trozos
% Variables de salida:
% ye evaluación de la función polinómica de los splines cúbicos en x
% ind trozo polinómico que ha evaluado
% Ejemplo.
% [y,ind]=SplinesCubicos([0 0.5 1 2 3 4 5 6 7 7.5 8],[0 8.9 11.8 15.5
15.7 15.7 15.7 13.6 9.6 5.8 0],0,0,5.5)

% Número de puntos en la tabla
n=length(X);

% Definimos el vector h de distancias entre las abscisas
h=diff(X,1);

% Inicializamos a cero los valores de la derivada segunda
z=zeros(1,n);
z(1)=z1;
z(n)=zn;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Cálculo de las derivadas segundas
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Definimos la matriz de coeficientes del sistema lineal
A=zeros(n-2,n-2);

if n>3
    % primera fila de la matriz
    A(1,1)=2*(h(1)+h(2));  A(1,2)=h(2);
    % filas intermedias
    for i=3:n-2
        A(i-1,i-2)=h(i-1);
        A(i-1,i-1)=2*(h(i-1)+h(i));
        A(i-1,i)=h(i);
    end
    % última fila
```

```
    A(n-2,n-3)=h(n-2);  A(n-2,n-2)=2*(h(n-1)+h(n-2));

elseif n==3
    % en este caso sólo hay una ecuación
    A(1,1)=2*(h(1)+h(2));

else
    display('Debe haber más de 2 puntos en la tabla');
    return
end


% Definimos el término independiente
b=zeros(n-2,1);

b(1)=6/h(2)*(Y(3)-Y(2))-6/h(1)*(Y(2)-Y(1))-h(1)*z(1);
for i=3:n-1
    b(i-1)=6/h(i)*(Y(i+1)-Y(i))-6/h(i-1)*(Y(i)-Y(i-1));
end
b(n-2)=b(n-2)-h(n-1)*z(n);


% Resolución del sistema
z(2:n-1)=A\b;

z

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Evaluación de los splines en x
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Encuentra el trozo polinómico a evaluar
ind=find((xe>X)==0);
ind=ind(1)-1;
syms x
% Aplica la fórmula para calcular S_ind(x)
y=z(ind+1)/(6*h(ind))*(x-X(ind))^3+z(ind)/(6*h(ind))*(X(ind+1)-x)^3+
...
    (Y(ind+1)/h(ind)-z(ind+1)*h(ind)/6)*(x-X(ind))+(Y(ind)/h(ind)-
z(ind)*h(ind)/6)*(X(ind+1)-x);
ye=subs(y,xe);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Polinomios S_i(x)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



for i=1:n-1

   S(i)=z(i+1)/(6*h(i))*(x-X(i))^3+z(i)/(6*h(i))*(X(i+1)-
x)^3+(Y(i+1)/h(i)-z(i+1)*h(i)/6)*(x-X(i))+(Y(i)/h(i)-
z(i)*h(i)/6)*(X(i+1)-x);
   S1(i)=diff(S(i),1);
   S2(i)=diff(S(i),2);
end
```

```
S(1)

S
S1
S2

% dibujo del polinomio de Taylor y de los puntos iniciales
dx=(X(1)-X(n))/10;
    A=X(1)-dx;
    B=X(n)+dx;
for i=1:n-1
    if i==1
        h(1)=ezplot(S(1),[A,X(i+1)]);
    elseif i==n
        ezplot(S(n),[X(i),B]);
    else
       h(i)=ezplot(S(i),[X(i),X(i+1)]);
    end
end

% puntos iniciales
hold on
p=plot(X,Y,'ro','MarKerSize',10);
```

Capitulo V.

Aproximación de la primera derivada.

```
function primera_derivada(f,c,h,num)

% Esta función aplica las reglas de aproximación de la primera
derivada
% usuales y calcula el error cometido en la aproximación, que se llama
% error de discretización
% primera_derivada_error_discretizacion(f,c,h)
% Variables de entrada:
% f función como cadena de caracteres
% c punto donde queremos aproximar la derivada
% h paso de discretización
% num número de puntos a utilizar
% Ejemplo:
%  primera_derivada('cos(x)',1,1/10,3)

% Derivada exacta en c
df=diff(f);
dfc=subs(df,c)
fprintf('Puntos utilizados %d\n',num);
if num==2
% Dos puntos

% Aproximación por la derivada progresiva

dpro=(subs(f,c+h)-subs(f,c))/h;
fprintf('Valor aproximado de forma progresiva %f\n',dpro);
```

```
fprintf('Error de discretización en la fórmula progresiva = %f\n',dfc-
dpro);

% Aproximación por la derivada regresiva

dreg=(subs(f,c)-subs(f,c-h))/h;
fprintf('Valor aproximado de forma regresiva %f\n',dreg);
fprintf('Error de discretización en la fórmula regresiva = %f\n',dfc-
dreg);

% Aproximación por la derivada centrada

dcen=(subs(f,c+h)-subs(f,c-h))/(2*h);
fprintf('Valor aproximado de forma centrada %f\n',dcen);
fprintf('Error de discretización en la fórmula centrada = %f\n',dfc-
dcen);

elseif num==3
% Tres puntos

%forma progresiva
dpro=((-3)*subs(f,c)+4*subs(f,c+h)-subs(f,c+2*h))/2/h;
fprintf('Valor aproximado de forma progresiva %f\n',dpro);
fprintf('Error de discretización en la fórmula progresiva = %f\n',dfc-
dpro);

%forma regresiva.
dreg=(3*subs(f,c)-4*subs(f,c-h)+subs(f,c-2*h))/2/h;
fprintf('Valor aproximado de forma regresiva %f\n',dreg);
fprintf('Error de discretización en la fórmula regresiva = %f\n',dfc-
dreg);

elseif num==4
%Cuatro puntos.

%forma centrada.
dcen=(subs(f,c-2*h)-8*subs(f,c-h)+8*subs(f,c+h)-subs(f,c+2*h))/12/h;
fprintf('Valor aproximado de forma centrada %f\n',dcen);
fprintf('Error de discretización en la fórmula centrada = %f\n',dfc-
dcen);

elseif num<2
    error('Debe utilizar al menos dos puntos');
elseif num>4
    error('Las expresiones utilizadas no admiten más de cuatro
puntos');
end
```

Aproximación de la segunda derivada.

```
function segunda_derivada(f,c,h,num)

% Esta función aplica distintas reglas useales la regla de
aproximación de la segunda derivada
% según el número de puntos que deseemos utilizar y calcula el error
cometido en la
% aproximación, que se llama error de discretización
```

```
% segunda_derivada_error_discretizacion(f,c,h)
% Variables de entrada:
% f función como cadena de caracteres
% c punto donde queremos aproximar la derivada
% h paso de discretización
% num número de puntos a utilizar
% Ejemplo:
% segunda_derivada('cos(x)',1,1/10,4)

% Derivada exacta en c
d2f=diff(f,2);
d2fc=subs(d2f,c)
fprintf('Numero de puntos utilizados %d\n',num);
%Tres puntos.
if num==3
    d2apro=(subs(f,c+h)-2*subs(f,c)+subs(f,c-h))/h^2;
    fprintf('Valor aproximado de la segunda derivada=%f\n',d2apro);
    error=d2fc-d2apro;
    if error>0
        fprintf('Aproximación por defecto\n');
    elseif error<0
        fprintf('Aproximación por exceso\n');
    end

    fprintf('Error de discretización en la fórmula para la segunda
derivada = %f\n',error);
%Cuatro puntos
elseif num==4

    fprintf('Aproximación progresiva\n');
    %Progresiva
    d2pro=(2*subs(f,c)-5*subs(f,c+h)+4*subs(f,c+2*h)-
subs(f,c+3*h))/(h^2);
    fprintf('Valor aproximado de la segunda derivada=%f\n',d2pro);
    error=d2fc-d2pro;
    if error>0
        fprintf('Aproximación por defecto\n');
    elseif error<0
        fprintf('Aproximación por exceso\n');
    end

    fprintf('Error de discretización en la fórmula para la segunda
derivada = %f\n',error);

    %Regresiva

    fprintf('Aproximación regresiva\n');

    d2reg=(2*subs(f,c)-5*subs(f,c-h)+4*subs(f,c-2*h)-subs(f,c-
3*h))/(h^2);
    fprintf('Valor aproximado de la segunda derivada=%f\n',d2reg);
    error=d2fc-d2reg;
    if error>0
        fprintf('Aproximación por defecto\n');
    elseif error<0
        fprintf('Aproximación por exceso\n');
    end

    fprintf('Error de discretización en la fórmula para la segunda
derivada = %f\n',error);
```

```
% Cinco puntos

elseif num==5
    d2cen=((-1)*subs(f,c-2*h)+16*subs(f,c-h)-
30*subs(f,c)+16*subs(f,c+h)-subs(f,c+2*h))/(12*h^2);
    fprintf('Valor aproximado de la segunda derivada=%f\n',d2cen);
    error=d2fc-d2cen;
    if error>0
        fprintf('Aproximación por defecto\n');
    elseif error<0
        fprintf('Aproximación por exceso\n');
    end
    fprintf('Error de discretización en la fórmula para la segunda
derivada = %f\n',error);

elseif num<3
    error('Necesita utilizar al menos 3 puntos');
elseif num>5
    error('Las expresiones utilizadas no admiten más de cinco
puntos');
end
```

Aproximación de la tercera derivada.

```
function tercera_derivada(f,c,h)

% Esta función devuelve la aproximación de la tercera derivada de una
% función f en un punto c, utilizando cuatro puntos centrados al punto
c y
% calcula el error cometido, que es un error de discretización.
% Variables de entrada.
% f función de la que queremos aproximar su tercera derivada.
% c punto en el que queremos aproximar la tercera derivada.
% h paso de discretización.
% Ejemplo.
% tercera_derivada('cos(x)',1,1/10)


% derivada exacta en c
d3f=diff(f,3);
d3fc=subs(d3f,c)

% Aproximación
d3apro=(-subs(f,c-2*h)+2*subs(f,c-h)-
2*subs(f,c+h)+subs(f,c+2*h))/(2*h^3);
fprintf('Valor aproximado de la tercera derivada %f\n',d3apro);
error=d3fc-d3apro;
    if error>0
        fprintf('Aproximación por defecto\n');
    elseif error<0
        fprintf('Aproximación por exceso\n');
    end

    fprintf('Error de discretización en la fórmula para la tercera
derivada = %f\n',error);
```

Integración por trapecios.

```
function [int]=trapecios(f,h,a,b)

% Integración numérica con el método de los trapecios
% [int]=trapecios(f,M,a,b)
% Datos de entrada
%f función integrando que puede ser pasada como una cadena de
caracteres
%   o como un vector que contiene los valores numéricos de f evaluada
en
%   los nodos
% h distancia entre valores
% a,b extremos del intervalo
% Datos de salida
% int valor de la aproximación a la integral
% Ejemplo.
% [int]=trapecios([0 5.33 6.08 6.251 6.724],1.681,0,6.724)

% calculamos la longitud del intervalo

M=(b-a)/h;

 % inicialización de variables
  I=0;
if ischar(f)==1
  % calculo de los nodos
   x(1)=a;
   x(M+1)=b;


   for i=2:M
      x(i)=x(i-1)+h;
   end

% Regla de trapecios

 % denotaremos por E a la suma de los ordenadas en los extremos
  E=subs(f,x(1))+subs(f,x(M+1));

 % denotaremos por I a la suma de las ordenadas intermedias del método
  for i=2:M
     I=I+subs(f,x(i));
  end



else
    M=length(f);

    % Regla de trapecios

 % denotaremos por E a la suma de los ordenadas en los extremos
  E=f(1)+f(M);

 % denotaremos por I a la suma de las ordenadas intermedias del método
  for i=2:M
```

```
      I=I+f(i);
  end
end


 % Calculando el valor de la aproximación
  int=h*((E/2)+I);
```

Integración por Simpson.

```
function [int]=simpson(f,h,a,b)

% Aplicamos la regla de Simpson para hacer integración numérica.
% [int]=simpson(f,n,a,b)
% Datos de entrada
% f función integrando que puede ser pasada como una cadena de
caracteres
%   o como un vector que contiene los valores numéricos de f evaluada
en
%   los nodos
% h es valor de la distancia que separan los valores en abscisas
% a,b extremos del intervalo
% Datos de salida
% int valor de la aproximación a la integral
% Ejemplo.
% [int]=simpson([0 5.33 6.08 6.251 6.724],1.681,0,6.724)


% calculamos el número de veces a aplicar Simpson.
 n=(b-a)/(h);

 % inicialización de variables
  Ii=0;
  Ip=0;
if ischar(f)==1
  % calculo de los nodos
   x(1)=a;
   x(n+1)=b;

   for i=2:n
      x(i)=x(1)+(i-1)*h;
   end

% Regla de Simpson



 % denotaremos por E a la suma de los valores de los extremos
  E=subs(f,x(1))+subs(f,x(n+1));

 % denotaremos por Ii a la suma de las ordenadas intermedias impares
(pares en Matlab) en el método
  for i=2:2:n-1
     Ii=Ii+subs(f,x(i));
  end
```

```
 % denotaremos por Ii a la suma de las ordenadas intermedias pares
(impares en Matlab) en el método
  for i=3:2:n-2
     Ip=Ip+subs(f,x(i));
  end

else
  n=length(f);

  % denotaremos por E a la suma de los valores de los extremos
  E=f(1)+f(n);

 % denotaremos por Ii a la suma de las ordenadas intermedias impares
(pares en Matlab) en el método
  for i=2:2:n-1
     Ii=Ii+f(i);
  end

 % denotaremos por Ii a la suma de las ordenadas intermedias pares
(impares en Matlab) en el método
  for i=3:2:n-2
     Ip=Ip+f(i);
  end

end

 % Calculando el valor de la aproximación
  int=(h/3)*(E+4*Ii+2*Ip);
```

Integración por Simpson con intervalos divididos.

```
function[R]=Simpson_dividido(f,H,xT)

% Esta función devuelve la integral de una función en un intervalo
calculada
% mediante el método de Simpson compuesto, con la posibilidad de
distinguir
% varios intervalos con equiespaciados distintos en cada una de las
fronteras.
% [R]=Simpson_dividido(f,H,xT)
% Variables de entrada
% f función integrando que puede ser pasada como una cadena de
caracteres
%   o como un vector que contiene los valores numéricos de f evaluada
en
%   los nodos
% H es el vector que contiene los distintos equiespaciodos para cada
uno de
% los intervalos.
% xT es un vector que contiene las abscisas x donde empiezan cada uno
de los
%    trozos. Sirve para calcular la longitud de los distintos
subintervalos
% Variables de salida
% int es el valor total de la integral realizada
% Ejemplo 1.
% Lpp=100 m, longitud total de la eslora
```

```
% [R]=Simpson_dividido([0 2.4 5.6 9.1 10 10.1 10.1 10.1 9.9 8.8 4.9
2.4 0],[5 10 5],[0 10 90 100])
% Ejemplo 2.
% [R]=Simpson_dividido('sin(x)',[0.005 0.01 0.05],[0 pi/6 5*pi/6 pi])


% número de trozos en que se parte el intervalo total
n=length(H);

% Calculamos la longitud de cada trozo
L=diff(xT);

% int guarda los valores de la integral aproximada en cada trozo
int=zeros(1,n);


if ischar(f)==0

    % Va acumulando el número de puntos hasta el principio de
    % cada bloque
    indT=0;

    for i=1:n

        M(i)=round(L(i)/H(i))+1;
        h(i)=L(i)/(M(i)-1);

        if rem(M(i),2)==1

            % Primera regla de Simpson.
            I=f(indT+1)+f(indT+M(i));
            for j=2:2:M(i)-1
                I=I+4*f(indT+j);
            end
            for j=3:2:M(i)-2
                I=I+2*f(indT+j);
            end
            int(i)=(h(i)/3)*I;

        elseif rem(M(i),2)==0

            % Aplicamos la primera regla hasta M-3
            I=f(indT+1)+f(indT+M(i)-3);
            for j=2:2:M(i)-4
                I=I+4*f(indT+j);
            end
            for j=3:2:M(i)-5
                I=I+2*f(indT+j);
            end
            I=I+(9/8)*f(indT+M(i)-3)+(27/8)*f(indT+M(i)-
2)+(27/8)*f(indT+M(i)-1)+(9/8)*f(indT+M(i));
            int(i)=(h(i)/3)*I;
        end

        indT=indT+M(i)-1;
    end

elseif ischar(f)==1
```

```
    for i=1:n

        M(i)=round(L(i)/H(i))+1;
        h(i)=L(i)/(M(i)-1);

        if rem(M(i),2)==1


            % Primera regla de Simpson.
            I=subs(f,xT(i))+subs(f,xT(i)+(M(i)-1)*h(i));
            for j=2:2:M(i)-1
                I=I+4*subs(f,xT(i)+(j-1)*h(i));
            end
            for j=3:2:M(i)-2
                I=I+2*subs(f,xT(i)+(j-1)*h(i));
            end
            int(i)=(h(i)/3)*I;

        elseif rem(M(i),2)==0

            % Aplicamos la primera regla hasta M-3
            I=subs(f,xT(i))+subs(f,xT(i)+(M(i)-4)*h(i));
            for j=2:2:M(i)-4
                I=I+4*subs(f,xT(i)+(j-1)*h(i));
            end
            for j=3:2:M(i)-5
                I=I+2*subs(f,xT(i)+(j-1)*h(i));
            end
            I=I+(9/8)*subs(f,xT(i)+(M(i)-
4)*h(i))+(27/8)*subs(f,xT(i)+(M(i)-3)*h(i))+(27/8)*subs(f,xT(i)+(M(i)-
2)*h(i))+(9/8)*subs(f,xT(i)+(M(i)-1)*h(i));
            int(i)=(h(i)/3)*I;
        end


    end
end


R=0;
%Cálculo final de la integral.
for i=1:n
    R=R+int(i);
end
```

Capitulo VI.

Método de Euler explícito.

```
function [T,Y,Ye,e]=euler_explicito(f,a,b,ya,h)

% Metodo de Euler explícito para el problema de valores iniciales
% y'=f(t,y), y(a)=y0
% [T,Y,Ye]=euler_explicito(f,a,b,ya,M);
% Datos de entrada:
% f cadena de caracteres identificando el termino fuente
% a y b son los extremos del intervalo
% ya es la condición inicial
% h es el valor de paso en cada iteración
% Datos de salida
% T es el vector de las abscisas de la solución
% Y es el vector de las ordenadas
% Ye valor de la solución exacta
% e diferencia entre la solución exacta y la aproximada en cada una de
las
% iteraciones
% [T,Y,Ye,e]=euler_explicito('(t-y)/2',0,3,0,1/2)



syms t y;

M=(b-a)/h;
T=zeros(1,M+1);
Y=zeros(1,M+1);

T=a:h:b;
Y(1)=ya;
for j=1:M
    Y(j+1)=Y(j)+h*subs(f,{t,y},{T(j),Y(j)});
end

% Código para comparar con la solucion exacta del problema

ecuacion=['Dy=',f];
condicion_inicial=['y(',num2str(a),')=',num2str(ya)];
S=dsolve(ecuacion,condicion_inicial);
if isempty(S)==0

Ye(1)=ya;
for j=2:M+1
    Ye(j)=subs(S,T(j));
end


e=(Ye-Y);

% Código para pintar las soluciones aproximadas y reales con
% colores diferentes

h1=plot(T,Y,'o');
```

```
hold on
h2=ezplot(S,[a,b]);
h3=plot(T,Ye,'ro');
legend([h1,h2,h3],'Aproximación en los puntos','Función exacta','Valor
exacto en los puntos');

else
    e=[];
    Ye=[];
    fprintf('No existe una expresión exacta para la solucón');
    h1=plot(T,Y,'o');
    legend([h1],'Aproximación en los puntos');
end
```

Método de Runge-Kutta de orden 4.

```
function [Y,Ye,e]=RKCuatro(f,a,y0,h,n)

% Programa para la resolución numérica de EDOs
% mediante el método de Runge-Kutta Cuatro
% y(n+1)=y(n)+1/6*(k1+2k2+2k3+k4)
% k1=h*f(t(n),y(n))
% k2=h*f(t(n)+1/2h,y(n)+1/2k1)
% k3=h*f(t(n)+1/2h,y(n)+1/2k2)
% k4=h*f(t(n)+h,y(n)+k3)
% El  problema a estudio es
% y'(t)= f(t,y)
% y(a)=y0
% [Y,Ye]=RKCuatro(f,a,y0,h,n)
% Datos de Entrada
% f cadena de caracteres identificando la función
% a tiempo inicial
% y0 condición inicial
% h paso temporal
% n número de pasos temporales a realizar
% Datos de Salida
% Y aproximación a la solución para los tiempos desde a hasta a+nh con
paso
%   h
% Ye solución discretizada para los tiempos desde a hasta a+nh con
paso
%   h
% e error en la aproximación para los tiempos desde a hasta a+nh con
paso h
% Ejemplo:
% Resolver y'(t)= exp(-2*t)-2*y,  y(0)=1/10 con paso h=0.8, y haced 2
pasos de tiempo
% [Y,Ye,e]=RKCuatro('exp(-2*t)-2*y',0,1/10,0.8,2)
% En esta ecuación podemos apreciar el error cometido



syms t y

tiempo(1)=a;  % tiempo inicial
```

```
% Inicialización  y=y0
Y(1)=y0;

% bucle que realiza los sucesivos pasos temporales

for i=1:n
    % siguiente paso temporal
    tiempo(i+1)=a+i*h;
    % cálculo de k1, k2, k3, k4
    k1=h*subs(f,{t,y},{tiempo(i),Y(i)});
    k2=h*subs(f,{t,y},{tiempo(i)+h/2,Y(i)+k1/2});
    k3=h*subs(f,{t,y},{tiempo(i)+h/2,Y(i)+k2/2});
    k4=h*subs(f,{t,y},{tiempo(i)+h,Y(i)+k3});
    % aproximación de Runge-Kutta 4
    Y(i+1)=Y(i)+1/6*(k1+2*k2+2*k3+k4);

end


% Código para comparar con la solucion exacta del problema

ecuacion=['Dy=',f];
condicion_inicial=['y(',num2str(a),')=',num2str(y0)];
S=dsolve(ecuacion,condicion_inicial);
if isempty(S)==0
Ye(1)=y0;
for j=2:n+1
    Ye(j)=subs(S,tiempo(j));
end



% Imprimimos en pantalla los diferentes tiempos, la solución exacta,
la
% aproximada, y el error
e=Ye'-Y';


% Dibujamos en pantalla la solución aproximada y la real con
% colores diferentes

h1=plot(tiempo,Y,'o');
hold on
h2=ezplot(S,[a,a+h*n]);
h3=plot(tiempo,Ye,'ro');
legend([h1,h2,h3],'Aproximación en los puntos','Función exacta','Valor
exacto en los puntos');
else
    Ye=[];
    frpintf('No existe una expresión exacta para la solución');
    h1=plot(tiempo,Y,'o');
    legend([h1],'Aproximación en los puntos');
end
```

Método de Euler para sistemas.

```
function [T,X,Y,Xe,Ye]=euler_explicito_sistemas(f,g,a,b,xa,ya,h)

%function
[T,X,Xe,Y,Ye]=euler_explicito_general_sistemas(f,g,a,b,xa,ya,h)
% Metodo de Euler explícito para sistemas. El problema de valores
iniciales
% x'=f(t,x,y), x(a)=xa
% y'=g(t,x,y), y(a)=ya
% [T,X,Y,Xe,Ye]=euler_explicito_general_sistemas(f,g,a,b,xa,ya,h);
% Datos de entrada:
% f cadena de caracteres identificando el termino fuente de la primera
% ecuacion
% g cadena de caracteres identificando el termino fuente de la segunda
% ecuacion
% a y b son los extremos del intervalo
% xa condicion inicial en la variable x
% ya es la condición inicial en la variable y
% h es el tamaño de paso de cada aproximación
% Datos de salida
% T es el vector de las abscisas de la solución
% X es el vector de las ordenadas en la variable x
% Xe valor de la solución exacta para la variable x
% Y es el vector de las ordenadas en la variable y
% Ye valor de la solución exacta para la variable y
% Ejemplo1:
% [T,X,Y,Xe,Ye]=euler_explicito_sistemas('x+y-
7','3*x+2*y+5',0,1,2,3,0.2)
% Ejemplo2, sin solución explícita.
% [T,X,Y,Xe,Ye]=euler_explicito_sistemas('x*y+t','x-t',0,1,0,1,0.2)


syms t x y;



M=(b-a)/h;
T=zeros(1,M+1);
X=zeros(1,M+1);
Y=zeros(1,M+1);

T=a:h:b;
X(1)=xa;
Y(1)=ya;
for j=1:M
    X(j+1)=X(j)+h*subs(f,{t,x,y},{T(j),X(j),Y(j)});
    Y(j+1)=Y(j)+h*subs(g,{t,x,y},{T(j),X(j),Y(j)});
end

% Código para comparar con la solucion exacta del problema

ecuacion1=['Dx=',f];
ecuacion2=['Dy=',g];
condicion_inicial1=['x(',num2str(a),')=',num2str(xa)];
condicion_inicial2=['y(',num2str(a),')=',num2str(ya)];
```

```
S=dsolve(ecuacion1,ecuacion2,condicion_inicial1,condicion_inicial2);
if isempty(S)==0

Xe(1)=xa;
Ye(1)=ya;
for j=2:M+1
    Xe(j)=subs(S(1).x,{t},{T(j)});
    Ye(j)=subs(S(1).y,{t},{T(j)});
end

% Calculamos los errores cometidos en cada aproximación.
ex=(Xe-X);
ey=(Ye-Y);


% Código para pintar las soluciones aproximadas y reales con
% colores diferentes

figure(1)
h1=plot(T,X,'o');
title('Variable x')
hold on

h2=plot(T,Xe,'ro');
legend([h1,h2],'Aproximación en los puntos','Valor exacto en los
puntos');


figure(2)
h4=plot(T,Y,'o');
title('Variable y')
hold on
h5=plot(T,Ye,'ro');
legend([h4,h5],'Aproximación en los puntos','Valor exacto en los
puntos');

else
    Xe=[];
    Ye=[];
    fprintf('No existe solución exacta para la ecuación diferencial');
    figure(1)
    h1=plot(T,X,'o');
    title('Variable x')
    legend([h1],'Aproximación de los puntos');

    figure(2)
    h4=plot(T,Y,'o');
    title('Variable y')
    legend([h4],'Aproximación de los puntos');
end
```

Método de Runge-Kutta orden 4 para sistemas.

```
function [X,Xe,ex,Y,Ye,ey]=RKCuatro_sistemas(f,g,a,x0,y0,h,n)

% Programa para la resolución numérica de sistemas de EDOs
% mediante el método de Runge-Kutta Cuatro
% x(n+1)=x(n)+1/6*(xk1+2xk2+2xk3+xk4)
% % xk1=h*f(t(n),x(n))
% % xk2=h*f(t(n)+1/2h,x(n)+1/2xk1)
% % xk3=h*f(t(n)+1/2h,x(n)+1/2xk2)
% % xk4=h*f(t(n)+h,x(n)+xk3)
% y(n+1)=y(n)+1/6*(k1+2k2+2k3+k4)
% % k1=h*f(t(n),y(n))
% % k2=h*f(t(n)+1/2h,y(n)+1/2k1)
% % k3=h*f(t(n)+1/2h,y(n)+1/2k2)
% % k4=h*f(t(n)+h,y(n)+k3)
% El  problema a estudio es
% x'(t)= f(t,y)
% y'(t)= g(t,y)
% x(a)=x0, y(a)=y0
% [X,Xe,Y,Ye,ex,ey]=RKCuatroSistemas(f,g,a,x0,y0,h,n)
% Datos de Entrada
% f cadena de caracteres identificando la función (x)
% g cadena de caracteres identificando la función (y)
% a tiempo inicial
% x0 condición inicial para x
% y0 condicion inicial para y
% h paso temporal
% n número de pasos temporales a realizar
% Datos de Salida
% X aproximación a la solución para tiempos desde a hasta a+nh con
paso h
% Y aproximación a la solución para tiempos desde a hasta a+nh con
paso h
% Xe solución discretizada para tiempos desde a hasta a+nh con paso h
% Ye solución discretizada para tiempos desde a hasta a+nh con paso h
% ex error en la aproximación para los tiempos desde a hasta a+nh con
paso h
% ey error en la aproximación para los tiempos desde a hasta a+nh con
paso h
% Ejemplo 1. Sistema sin solución explícita.
% Resolver x'=x*y+t, y'(t)=x-t, x(0)=0, y(0)=1 con paso h=0.1, y haced
2 pasos de tiempo
% [X,Xe,ex,Y,Ye,ey]=RKCuatro_sistemas('x*y+t','x-t',0,0,1,0.1,2)
% Ejemplo 2. Sistema con solución explicita.
% [X,Xe,ex,Y,Ye,ey]=RKCuatro_sistemas('x+y-7','3*x+2*y+5',0,2,3,1,3)



syms t y x


tiempo(1)=a;  % tiempo inicial

% Inicialización
Y(1)=y0;
X(1)=x0;
```

```
% bucle que realiza los sucesivos pasos temporales

for i=1:n
    % siguiente paso temporal
    tiempo(i+1)=a+i*h;

    % cálculo de k1, k2, k3, k4
    k1x=h*subs(f,{t,x,y},{tiempo(i),X(i),Y(i)});
    k1y=h*subs(g,{t,x,y},{tiempo(i),X(i),Y(i)});
    k2x=h*subs(f,{t,x,y},{tiempo(i)+h/2,X(i)+k1x/2,Y(i)+k1y/2});
    k2y=h*subs(g,{t,x,y},{tiempo(i)+h/2,X(i)+k1x/2,Y(i)+k1y/2});
    k3x=h*subs(f,{t,x,y},{tiempo(i)+h/2,X(i)+k2x/2,Y(i)+k2y/2});
    k3y=h*subs(g,{t,x,y},{tiempo(i)+h/2,X(i)+k2x/2,Y(i)+k2y/2});
    k4x=h*subs(f,{t,x,y},{tiempo(i)+h,X(i)+k3x,Y(i)+k3y});
    k4y=h*subs(g,{t,x,y},{tiempo(i)+h,X(i)+k3x,Y(i)+k3y});

    % aproximación de Runge-Kutta 4
    X(i+1)=X(i)+1/6*(k1x+2*k2x+2*k3x+k4x);
    Y(i+1)=Y(i)+1/6*(k1y+2*k2y+2*k3y+k4y);
end


% Código para comparar con la solucion exacta del problema

ecuacion1=['Dx=',f];
ecuacion2=['Dy=',g];
condicion_inicial1=['x(',num2str(a),')=',num2str(x0)];
condicion_inicial2=['y(',num2str(a),')=',num2str(y0)];

S=dsolve(ecuacion1,ecuacion2,condicion_inicial1,condicion_inicial2);
if isempty(S)==0

Xe(1)=x0;
Ye(1)=y0;
for i=2:n+1
    Xe(i)=subs(S(1).x,{t},{tiempo(i)});
    Ye(i)=subs(S(1).y,{t},{tiempo(i)});
end



% Error cometido
ex=abs(Xe-X);
ey=abs(Ye-Y);




% Dibujamos en pantalla la solución aproximada y la real con
% colores diferentes
figure(1)
h1=plot(tiempo,X,'o');
title('Variable x')
hold on
h2=plot(tiempo,Xe,'ro');
legend([h1,h2],'Aproximación obtenida','Valor exacto');
hold off
```

```
figure(2)
h3=plot(tiempo,Y,'o');
title('Variable y')
hold on
h4=plot(tiempo,Ye,'ro');
legend([h3,h4],'Aproximación obtenida','Valor exacto');
hold off
else
    Xe=[];
    Ye=[];
    ex=[];
    ey=[];
    fprintf('No existe solución exacta del problema')
    figure(1)
    h1=plot(tiempo,X,'o');
    title('Variable x')
    legend([h1],'Aproximación obtenida');

    figure(2)
    h3=plot(tiempo,Y,'o');
    title('Variable y')
    legend([h3],'Aproximación obtenida');
end
```

Bibliografía.

- Métodos numéricos con Matlab.3rd edición. John H. Mathews, Kurtis D.Fink.

- Cálculo Numérico (Teoría y problemas). Antonio Vigueras Campuzano. 3 de Octubre de 2014.

- Derivación e integración numéricas. Gabriel Soler López. 6 de Abril de 2004.

- Métodos numéricos para ecuaciones diferenciales. José S. Cánovas Peña. 18 de diciembre de 2009.

- Diseño y cálculo de estructuras navales. José Alfonso Martínez García. Curso 2012/2013.

- Apuntes Hidrostática y estabilidad del buque. Domingo L.García López.